Oracle® Big Data Connectors

User's Guide

Release 2 (2.1)

E36961-05

July 2013

Describes installation and use of Oracle Big Data Connectors: Oracle SQL Connector for Hadoop Distributed File System, Oracle Loader for Hadoop, Oracle Data Integrator Application Adapter for Hadoop, and Oracle R Connector for Hadoop.

ORACLE®

Oracle Big Data Connectors User's Guide, Release 2 (2.1)

E36961-05

Contents

2 Oracle SQL Connector for Hadoop Distributed File System

3 Oracle Loader for Hadoop

4 Oracle Data Integrator Application Adapter for Hadoop

5 Oracle R Connector for Hadoop

6 ORCH Library Reference

Index

Preface

The *Oracle Big Data Connectors User's Guide* describes how to install and use Oracle Big Data Connectors:

- Oracle Loader for Hadoop
- Oracle SQL Connector for Hadoop Distributed File System
- Oracle Data Integrator Application Adapter for Hadoop
- Oracle R Connector for Hadoop

Audience

This document is intended for users of Oracle Big Data Connectors, including the following:

- Application developers
- Java programmers
- System administrators
- Database administrators

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Loader for Hadoop Java API Reference*
- *Oracle Fusion Middleware Application Adapters Guide for Oracle Data Integrator*

Text Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Syntax Conventions

The syntax in Chapter 2 is presented in a simple variation of Backus-Naur Form (BNF) that uses the following symbols and conventions:

Symbol or Convention	Description
[ ]	Brackets enclose optional items.
{ }	Braces enclose a choice of items, only one of which is required.
|	A vertical bar separates alternatives within brackets or braces.
...	Ellipses indicate that the preceding syntactic element can be repeated.
delimiters	Delimiters other than brackets, braces, and vertical bars must be entered as shown.

Changes in This Release for Oracle Big Data Connectors User's Guide

This preface lists changes in the *Oracle Big Data Connectors User's Guide*.

Changes in Oracle Big Data Connectors Release 2 (2.0)

The following are changes in *Oracle Big Data Connectors User's Guide* for Oracle Big Data Connectors Release 2 (2.0).

New Features

Oracle Big Data Connectors support Cloudera's Distribution including Apache Hadoop version 4 (CDH4). For other supported platforms, see the individual connectors in Chapter 1.

The name of Oracle Direct Connector for Hadoop Distributed File System changed to Oracle SQL Connector for Hadoop Distributed File System.

- **Oracle SQL Connector for Hadoop Distributed File System**
  - Automatic creation of Oracle Database external tables from Hive tables, Data Pump files, or delimited text files.
  - Management of location files.

  See Chapter 2.
- **Oracle Loader for Hadoop**
  - Support for Sockets Direct Protocol (SDP) for direct path loads
  - Support for secondary sort on user-specified columns
  - New input formats for regular expressions and Oracle NoSQL Database. The Avro record InputFormat is supported code instead of sample code.
  - Simplified date format specification
  - New reject limit threshold
  - Improved job reporting and diagnostics

  See Chapter 3.
- **Oracle Data Integrator Application Adapter for Hadoop**
  - Uses Oracle SQL Connector for HDFS or Oracle Loader for Hadoop to load data from Hadoop into an Oracle database.
- **Oracle R Connector for Hadoop**

Several analytic algorithms are now available: linear regression, neural networks for prediction, matrix completion using low rank matrix factorization, clustering, and non-negative matrix factorization.

Oracle R Connector for Hadoop supports Hive data sources in addition to HDFS files.

Oracle R Connector for Hadoop can move data between HDFS and Oracle Database. Oracle R Enterprise is not required for this basic transfer of data.

The following functions are new in this release:

```
as.ore.*
hadoop.jobs
hdfs.head
hdfs.tail
is.ore.*
orch.connected
orch.dbg.lasterr
orch.evaluate
orch.export.fit
orch.lm
orch.lmf
orch.neural
orch.nmf
orch.nmf.NMFalgo
orch.temp.path
ore.*
predict.orch.lm
print.summary.orch.lm
summary.orch.lm
```

See Chapter 5.

Deprecated Features

The following features are deprecated in this release, and may be desupported in a future release:

- **Oracle SQL Connector for Hadoop Distributed File System**
  - **Location file format (version 1)**: Existing external tables with content published using Oracle Direct Connector for HDFS version 1 must be republished using Oracle SQL Connector for HDFS version 2, because of incompatible changes to the location file format.

    When Oracle SQL Connector for HDFS creates new location files, it does not delete the old location files.

    See Chapter 2.
  - **`oracle.hadoop.hdfs.exttab` namespace (version 1)**: Oracle SQL Connector for HDFS uses the following new namespaces for all configuration properties:
    * `oracle.hadoop.connection`: Oracle Database connection and wallet properties
    * `oracle.hadoop.exttab`: All other properties

    See Chapter 2.
  - **HDFS_BIN_PATH directory**: The preprocessor directory name is now OSCH_BIN_PATH.

See "Oracle SQL Connector for Hadoop Distributed File System Setup" on page 1-4.

- **Oracle R Connector for Hadoop**
  - `keyval`: Use `orch.keyval` to generate key-value pairs.
  - `orch.reconnect`: Use `orch.connect` to reconnect using a connection object returned by `orch.dbcon`.

Desupported Features

The following features are no longer supported by Oracle.

- Oracle Loader for Hadoop
  - `oracle.hadoop.loader.configuredCounters`

    See Chapter 3.

Other Changes

The following are additional changes in the release:

- **Oracle Loader for Hadoop**

  The installation zip archive now contains two kits:

  - oraloader-2.0.0-2.x86_64.zip for CDH4
  - oraloader-2.0.0-1.x86_64.zip for Apache Hadoop 0.20.2 and CDH3

  See "Oracle Loader for Hadoop Setup" on page 1-12.

1

Getting Started with Oracle Big Data Connectors

This chapter describes the Oracle Big Data Connectors, provides installation instructions, and identifies the permissions needed for users to access the connectors.

This chapter contains the following sections:

About Oracle Big Data Connectors

Oracle Big Data Connectors facilitate data access between data stored in a Hadoop cluster and Oracle Database. They can be licensed for use on either Oracle Big Data Appliance or a Hadoop cluster running on commodity hardware.

These are the connectors:

- **Oracle SQL Connector for Hadoop Distributed File System (previously Oracle Direct Connector for HDFS)**: Enables an Oracle external table to access data stored in Hadoop Distributed File System (HDFS) files or a Hive table. The data can remain in HDFS or the Hive table, or it can be loaded into an Oracle database. Oracle SQL Connector for HDFS is a command-line utility that accepts generic command line arguments supported by the org.apache.hadoop.util.Tool interface. It also provides a preprocessor for Oracle external tables.
- **Oracle Loader for Hadoop**: Provides an efficient and high-performance loader for fast movement of data from a Hadoop cluster into a table in an Oracle database. Oracle Loader for Hadoop prepartitions the data if necessary and transforms it into a database-ready format. It optionally sorts records by primary key or user-defined columns before loading the data or creating output files. Oracle Loader for Hadoop is a MapReduce application that is invoked as a command-line utility. It accepts the generic command-line options that are supported by the org.apache.hadoop.util.Tool interface.

- **Oracle Data Integrator Application Adapter for Hadoop**: Extracts, transforms, and loads data from a Hadoop cluster into tables in an Oracle database, as defined using a graphical user interface.
- **Oracle R Connector for Hadoop**: Provides an interface between a local R environment, Oracle Database, and Hadoop, allowing speed-of-thought, interactive analysis on all three platforms. Oracle R Connector for Hadoop is designed to work independently, but if the enterprise data for your analysis is also stored in Oracle Database, then the full power of this connector is achieved when it is used with Oracle R Enterprise.

Individual connectors may require that software components be installed in Oracle Database and the Hadoop cluster. Users may also need additional access privileges in Oracle Database.

> **See Also:** My Oracle Support Information Center: Big Data Connectors (ID 1487399.2) and its related information centers.

Big Data Concepts and Technologies

Enterprises are seeing large amounts of data coming from multiple sources. Click-stream data in web logs, GPS tracking information, data from retail operations, sensor data, and multimedia streams are just a few examples of vast amounts of data that can be of tremendous value to an enterprise if analyzed. The unstructured and semi-structured information provided by raw data feeds is of little value in and of itself. The data needs to be processed to extract information of real value, which can then be stored and managed in the database. Analytics of this data along with the structured data in the database can provide new insights into the data and lead to substantial business benefits.

What is MapReduce?

MapReduce is a parallel programming model for processing data on a distributed system. It can process vast amounts of data in a timely manner and can scale linearly. It is particularly effective as a mechanism for batch processing of unstructured and semi-structured data. MapReduce abstracts lower level operations into computations over a set of keys and values.

A simplified definition of a MapReduce job is the successive alternation of two phases, the map phase and the reduce phase. Each map phase applies a transform function over each record in the input data to produce a set of records expressed as key-value pairs. The output from the map phase is input to the reduce phase. In the reduce phase, the map output records are sorted into key-value sets so that all records in a set have the same key value. A reducer function is applied to all the records in a set and a set of output records are produced as key-value pairs. The map phase is logically run in parallel over each record while the reduce phase is run in parallel over all key values.

> **Note:** Oracle Big Data Connectors do not support the Yet Another Resource Negotiator (YARN) implementation of MapReduce.

What is Apache Hadoop?

Apache Hadoop is the software framework for the development and deployment of data processing jobs based on the MapReduce programming model. At the core, Hadoop provides a reliable shared storage and analysis system[1]. Analysis is provided

by MapReduce. Storage is provided by the Hadoop Distributed File System (HDFS), a shared storage system designed for MapReduce jobs.

The Hadoop ecosystem includes several other projects including Apache Avro, a data serialization system that is used by Oracle Loader for Hadoop.

Cloudera's Distribution including Apache Hadoop (CDH) is installed on Oracle Big Data Appliance. You can use Oracle Big Data Connectors on a Hadoop cluster running CDH or the equivalent Apache Hadoop components, as described in the setup instructions in this chapter.

See Also:

- For conceptual information about the Hadoop technologies, the following third-party publication:

  Hadoop: The Definitive Guide, Third Edition by Tom White (O'Reilly Media Inc., 2012, ISBN: 978-1449311520).

- For information about Cloudera's Distribution including Apache Hadoop (CDH4), the Oracle Cloudera website at

  http://oracle.cloudera.com/

- For information about Apache Hadoop, the website at

  http://hadoop.apache.org/

Downloading the Oracle Big Data Connectors Software

You can download Oracle Big Data Connectors from Oracle Technology Network or Oracle Software Delivery Cloud.

To download from Oracle Technology Network:

1. Use any browser to visit this website:

   http://www.oracle.com/technetwork/bdc/big-data-connectors/downloads/index.html

2. Click the name of each connector to download a zip file containing the installation files.

To download from Oracle Software Delivery Cloud:

1. Use any browser to visit this website:

   https://edelivery.oracle.com/

2. Accept the Terms and Restrictions to see the Media Pack Search page.
3. Select the search terms:

   Select a Product Pack: Oracle Database

   Platform: Linux x86-64

4. Click **Go** to display a list of product packs.
5. Select Oracle Big Data Connectors Media Pack for Linux x86-64 (B65965-0*x*), and then click **Continue**.

[1] *Hadoop: The Definitive Guide, Third Edition* by Tom White (O'Reilly Media Inc., 2012, 978-1449311520).

6. Click **Download** for each connector to download a zip file containing the installation files.

Oracle SQL Connector for Hadoop Distributed File System Setup

You install and configure Oracle SQL Connector for Hadoop Distributed File System (HDFS) on the system where Oracle Database runs. If Hive tables are used as the data source, then you must also install and run Oracle SQL Connector for HDFS on the Hadoop cluster where Hive is installed.

Oracle SQL Connector for HDFS is installed already on Oracle Big Data Appliance if it was configured for Oracle Big Data Connectors.

This section contains the following topics:

Software Requirements

Oracle SQL Connector for HDFS requires the following software:

On the Hadoop cluster:

- Cloudera's Distribution including Apache Hadoop version 3 (CDH3) or version 4 (CDH4), or Apache Hadoop 1.0 (formerly 0.20.2).
- Java Development Kit (JDK) 1.6_08 or later. Consult the distributor of your Hadoop software (Cloudera or Apache) for the recommended version.
- Hive 0.7.0, 0.8.1, or 0.9.0 (required for Hive table access, otherwise optional)

These components are already installed on Oracle Big Data Appliance.

On the Oracle Database system:

- Oracle Database release 11*g* release 2 (11.2.0.2 or 11.2.0.3) for Linux.
- To support the Oracle Data Pump file format in Oracle Database release 11.2.0.2, an Oracle Database one-off patch. To download this patch, go to `http://support.oracle.com` and search for bug 14557588.

  Release 11.2.0.3 and later releases do not require this patch.
- The same version of Hadoop as your Hadoop cluster, either CDH3, CDH4, or Apache Hadoop 1.0.
- The same version of JDK as your Hadoop cluster.

Installing and Configuring a Hadoop Client

Oracle SQL Connector for HDFS works as a Hadoop client. You must install Hadoop on the Oracle Database system and minimally configure it for Hadoop client use only.

However, you do not need to perform a full configuration of Hadoop on the Oracle Database system to run MapReduce jobs for Oracle SQL Connector for HDFS.

To configure the Oracle Database system as a Hadoop client:

1. Install and configure the same version of CDH or Apache Hadoop on the Oracle Database system as on the Hadoop cluster. If you are using Oracle Big Data Appliance, then complete the procedures for providing remote client access in the *Oracle Big Data Appliance Software User's Guide*. Otherwise, follow the installation instructions provided by the distributor (Cloudera or Apache).

   Note: Do not start Hadoop on the Oracle Database system. If it is running, then Oracle SQL Connector for HDFS attempts to use it instead of the Hadoop cluster. Oracle SQL Connector for HDFS just uses the Hadoop JAR files and the configuration files from the Hadoop cluster on the Oracle Database system.

2. Ensure that Oracle Database has access to HDFS:

   a. Log in to the system where Oracle Database is running by using the Oracle Database account.

   b. Open a Bash shell and enter this command:

      ```
      hadoop fs -ls /user
      ```

      You might need to add the directory containing the Hadoop executable file to the `PATH` environment variable. The default path for CDH is /usr/bin.

      You should see the same list of directories that you see when you run the `hadoop fs` command directly on the Hadoop cluster. If not, then first ensure that the Hadoop cluster is up and running. If the problem persists, then you must correct the Hadoop client configuration so that Oracle Database has access to the Hadoop cluster file system.

The Oracle Database system is now ready for use as a Hadoop client. No other Hadoop configuration steps are needed.

Installing Oracle SQL Connector for HDFS

Follow this procedure to install Oracle SQL Connector for HDFS.

To install Oracle SQL Connector for HDFS on the Oracle Database system:

1. Download the zip file to a directory on the system where Oracle Database runs.
2. Unzip orahdfs-*version*.zip into a permanent directory:

```
$ unzip orahdfs-2.1.0.zip
Archive:  orahdfs-2.1.0.zip
   creating: orahdfs-2.1.0/
   creating: orahdfs-2.1.0/log/
   creating: orahdfs-2.1.0/doc/
  inflating: orahdfs-2.1.0/doc/README.txt
   creating: orahdfs-2.1.0/jlib/
  inflating: orahdfs-2.1.0/jlib/ojdbc6.jar
  inflating: orahdfs-2.1.0/jlib/orahdfs.jar
  inflating: orahdfs-2.1.0/jlib/oraclepki.jar
  inflating: orahdfs-2.1.0/jlib/osdt_cert.jar
  inflating: orahdfs-2.1.0/jlib/osdt_core.jar
   creating: orahdfs-2.1.0/bin/
```

```
  inflating: orahdfs-2.1.0/bin/hdfs_stream
```

The unzipped files have the structure shown in Example 1–1.

3. Open the orahdfs-2.1.0/bin/hdfs_stream Bash shell script in a text editor, and make the changes indicated by the comments in the script.

   The hdfs_stream script does not inherit any environment variable settings, and so they are set in the script if they are needed by Oracle SQL Connector for HDFS:

   - `PATH`: If the `hadoop` script is not in /usr/bin:bin (the path initially set in hdfs_stream), then add the Hadoop bin directory, such as /usr/lib/hadoop/bin.
   - `JAVA_HOME`: If Hadoop does not detect Java, then set this variable to the Java installation directory. For example, /usr/bin/java.
   - `OSCH_HOME`: If you moved the script from the orahdfs-*version*/bin subdirectory, then set this variable to the full path of the orahdfs-2.1.0 directory, which was created in Step 2. Otherwise, Oracle SQL Connector for HDFS detects its location automatically.

   See the comments in the script for more information about these environment variables.

   The hdfs_stream script is the preprocessor for the Oracle Database external table created by Oracle SQL Connector for HDFS.

   The hdfs_stream script is the preprocessor for the Oracle Database external table created by Oracle SQL Connector for HDFS.

   Note: The script does not inherit any environment variables set elsewhere, such as at the command line or in a shell initialization script (.bashrc).

4. Run hdfs_stream from the Oracle SQL Connector for HDFS installation directory. You should see this usage information:

   ```
   $ ./hdfs_stream
   Usage: hdfs_stream locationFile
   ```

   If you do not see the usage statement, then ensure that the operating system user that Oracle Database is running under has the following permissions:

   - Read and execute permissions on the hdfs_stream script:

     ```
     $ ls -l OSCH_HOME/bin/hdfs_stream
     -rwxr-xr-x 1 oracle oinstall Nov 27 15:51 hdfs_stream
     ```

   - Read permission on orahdfs.jar.

     ```
     $ ls -l OSCH_HOME/jlib/orahdfs.jar
     -rwxr-xr-x 1 oracle oinstall Nov 27 15:51 orahdfs.jar
     ```

   If you do not see these permissions, then enter a `chmod` command to fix them, for example:

   ```
   $ chmod 755 OSCH_HOME/bin/hdfs_stream
   ```

   In the previous commands, *OSCH_HOME* represents the Oracle SQL Connector for HDFS home directory.

5. Log in to Oracle Database and create a database directory for the orahdfs-*version*/bin directory where hdfs_stream resides. In this example, Oracle SQL Connector for HDFS is installed in /etc:

```
SQL> CREATE OR REPLACE DIRECTORY osch_bin_path AS '/etc/orahdfs-2.1.0/bin';
```

6. If you plan to access only data stored in HDFS and Data Pump format files, then you are done. If you also plan to access Hive tables, then you must also install software on the Hadoop cluster, as described in "Providing Support for Hive Tables" on page 1-8.

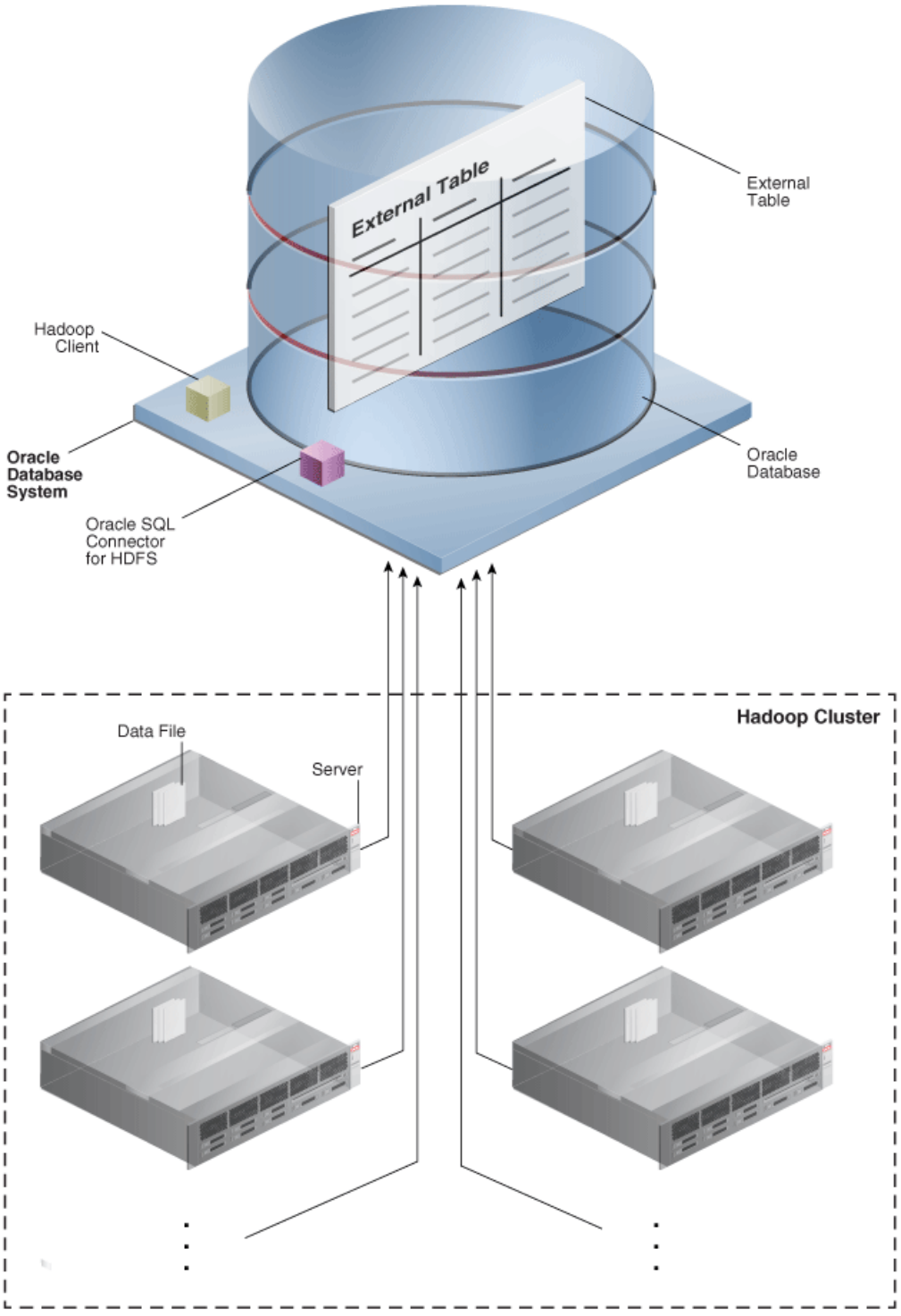
The unzipped files have the structure shown in Example 1–1.

Example 1–1 Structure of the orahdfs Directory

```
orahdfs-version
   bin/
      hdfs_stream
   jlib/
      orahdfs.jar
      osdt_core.jar
      osdt_cert.jar
      oraclepki.jar
      ojdbc6.jar
   log/
   doc/
      README.txt
```

Figure 1–1 illustrates shows the flow of data and the components locations.

Figure 1–1 Oracle SQL Connector for HDFS Installation for HDFS and Data Pump Files



Providing Support for Hive Tables

Complete the following procedure on the Hadoop cluster to support access to Hive tables. If you only plan to access HDFS and Data Pump format files, then you can omit this procedure.

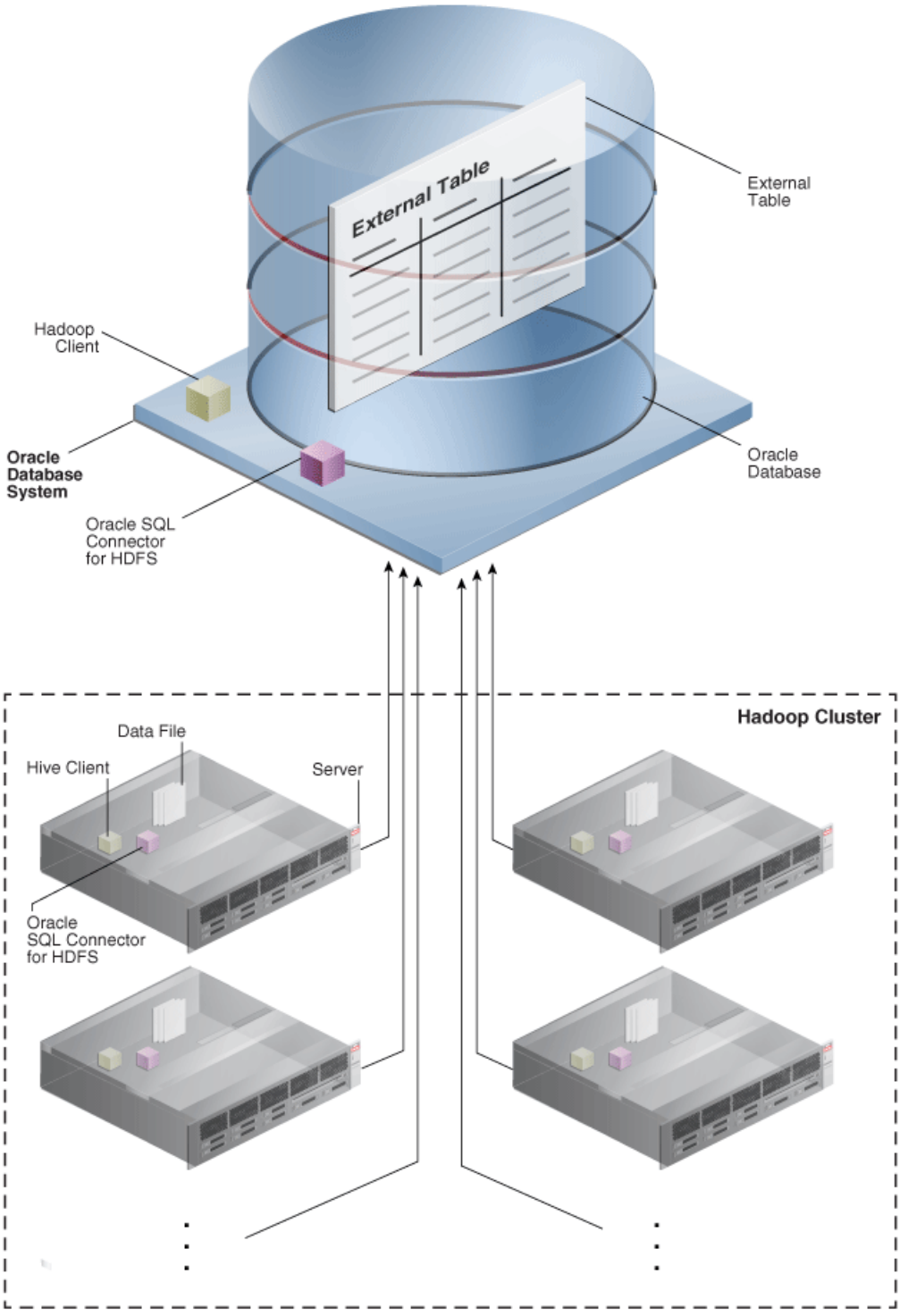
Oracle SQL Connector for HDFS should already be installed on Oracle Big Data Appliance. To verify the installation, ensure that /opt/oracle/orahdfs-2.1.0 exists.

To support Hive tables using a third-party Hadoop cluster:

1. Download or copy the orahdfs-*version*.zip file to a directory on a node in the Hadoop cluster where Hive is installed.
2. Unzip orahdfs-*version*.zip into a directory.
3. Repeat these steps for all nodes in the cluster.
4. Add the Hive JAR files and the Hive conf directory to the `HADOOP_CLASSPATH` environment variable. To avoid JAR conflicts among the various Hadoop products, Oracle recommends that you set `HADOOP_CLASSPATH` in the local shell initialization script of Oracle SQL Connector for HDFS users instead of making a global change to `HADOOP_CLASSPATH`. See "Setting Up User Accounts on the Hadoop Cluster" on page 1-11.

Figure 1–2 illustrates the flow of data and the component locations.

Figure 1–2 Oracle SQL Connector for HDFS Installation for Hive Tables



Granting User Privileges in Oracle Database

Oracle Database users require these privileges when using Oracle SQL Connector for HDFS to create external tables:

- `CREATE SESSION`

- CREATE TABLE
- EXECUTE on the UTL_FILE PL/SQL package
- READ and EXECUTE on the OSCH_BIN_PATH directory created during the installation of Oracle SQL Connector for HDFS. Do not grant write access to anyone. Grant EXECUTE only to those who intend to use Oracle SQL Connector for HDFS.
- READ and WRITE on a database directory for storing external tables, or the CREATE ANY DIRECTORY system privilege.
- A tablespace and quota for copying data into the Oracle database. Optional.

Example 1–2 shows the SQL commands granting these privileges to HDFSUSER.

Example 1–2 Granting Users Access to Oracle SQL Connector for HDFS

```
CONNECT / AS sysdba;
CREATE USER hdfsuser IDENTIFIED BY password
     DEFAULT TABLESPACE hdfsdata
     QUOTA UNLIMITED ON hdfsdata;
GRANT CREATE SESSION, CREATE TABLE TO hdfsuser;
GRANT EXECUTE ON sys.utl_file TO hdfsuser;
GRANT READ, EXECUTE ON DIRECTORY osch_bin_path TO hdfsuser;
GRANT READ, WRITE ON DIRECTORY external_table_dir TO hdfsuser;
```

Note: To query an external table that uses Oracle SQL Connector for HDFS, users only need the SELECT privilege on the table.

Setting Up User Accounts on the Oracle Database System

To create external tables for HDFS and Data Pump format files, users can log in to either the Oracle Database system or a node in the Hadoop cluster. For Hive access, user must log in to the cluster.

You can set up an account on the Oracle Database system the same as you would for any other operating system user. HADOOP_CLASSPATH must include *path*/orahdfs-2.1.0/jlib/*. You can add this setting to the shell profile as part of this installation procedure, or users can set it themselves. The following example alters HADOOP_CLASSPATH in the Bash shell where Oracle SQL Connector for HDFS is installed in /usr/bin:

```
export HADOOP_CLASSPATH="$HADOOP_CLASSPATH:/usr/bin/orahdfs-2.1.0/jlib/*
```

Setting Up User Accounts on the Hadoop Cluster

For Hive access, users must have a home directory and be able to log in to the nodes of a Hadoop cluster. After logging in to a node, users can run Oracle SQL Connector for HDFS to create Oracle Database external tables for all supported data sources.

To set up a Hadoop user account for Oracle SQL Connector for HDFS:

1. Create home directories in HDFS and in the local file system.

   For Oracle Big Data Appliance, see the instructions for creating new HDFS users in the *Oracle Big Data Appliance Software User's Guide*.

   For a Hadoop cluster running on commodity hardware, you can follow the same basic steps as you would on Oracle Big Data Appliance. The only difference is that you cannot use dcli to replicate your changes across all nodes.

2. Modify the shell configuration file (such as .bashrc for the Bash shell), to add the necessary JAR files to `HADOOP_CLASSPATH`:

   ```
   path/orahdfs-2.1.0/jlib/*
   /usr/lib/hive/lib/*
   /etc/hive/conf
   ```

   The following Bash command alters `HADOOP_CLASSPATH` on Oracle Big Data Appliance:

   ```
   export HADOOP_CLASSPATH="$HADOOP_
   CLASSPATH:/opt/oracle/orahdfs-2.1.0/jlib/*:/usr/lib/hive/lib/*:/etc/hive/conf"
   ```

3. Replicate the altered file across all nodes of the cluster. This command replicates the Bash shell configuration file across all nodes in Oracle Big Data Appliance:

   ```
   # dcli cp -f /home/user_name/.bashrc -d /home/user_name
   ```

   See Also: *Oracle Big Data Appliance Owner's Guide* for the full syntax of the `dcli` command.

Oracle Loader for Hadoop Setup

Before installing Oracle Loader for Hadoop, verify that you have the required software.

Software Requirements

Oracle Loader for Hadoop requires the following software:

- A target database system running one of the following:
  - Oracle Database 11*g* release 2 (11.2.0.2) with required patch
  - Oracle Database 11*g* release 2 (11.2.0.3)

  Note: To use Oracle Loader for Hadoop with Oracle Database 11*g* release 2 (11.2.0.2), you must first apply a one-off patch that addresses bug number 11897896. To access this patch, go to `http://support.oracle.com` and search for the bug number.

- Cloudera's Distribution including Apache Hadoop version 3 (CDH3) or version 4 (CDH4), or Apache Hadoop 1.0 (formerly 0.20.2).
- Hive 0.7.0, 0.8.1, 0.9.0, or 0.10.0 if you are loading data from Hive tables.

Installing Oracle Loader for Hadoop

Oracle Loader for Hadoop is packaged with the Oracle Database 11*g* release 2 client libraries and Oracle Instant Client libraries for connecting to Oracle Database 11.2.0.2 or 11.2.0.3.

To install Oracle Loader for Hadoop:

1. Unpack the content of oraloader-*version*.zip into a directory on your Hadoop cluster. This archive contains two archives:
   - oraloader-*version*-1.x86_64.zip: Use with CDH3 and Apache Hadoop 1.0

- oraloader-*version*-2.x86_64.zip: Use with CDH4

2. Unzip the appropriate archive into a directory on your Hadoop cluster.

   A directory named oraloader-*version-n* is created with the following subdirectories:

   ```
   doc
   jlib
   lib
   examples
   ```

3. Create a variable named `OLH_HOME` and set it to the installation directory.
4. Add `$OLH_HOME/jlib/*` to the `HADOOP_CLASSPATH` variable.
5. Add `$KVHOME/lib/kvstore.jar` to the `HADOOP_CLASSPATH` variable, if you are reading data from Oracle NoSQL Database Release 2.

Providing Support for Offline Database Mode

In a typical installation, Oracle Loader for Hadoop can connect to the Oracle Database system from the Hadoop cluster where the loader is installed. If this connection is impossible—for example, the two systems are located on distinct networks—then you can use Oracle Loader for Hadoop in offline database mode. You must install Oracle Loader for Hadoop on both the database system and on the Hadoop cluster. See "About the Modes of Operation" on page 3-2.

To support Oracle Loader for Hadoop in offline database mode:

1. Unpack the content of oraloader-*version*.zip into a directory on the Oracle Database system.
2. Unzip the same version of the software as you installed on the Hadoop cluster, either for CDH3 or CDH4.
3. Create a variable named `OLH_HOME` and set it to the installation directory. This example uses the Bash shell syntax:

   ```
   $ export OLH_HOME="/usr/bin/oraloader-2.1.0-h2/"
   ```

4. Add the Oracle Loader for Hadoop JAR files to the `CLASSPATH` environment variable. This example uses the Bash shell syntax:

   ```
   $ export CLASSPATH=$CLASSPATH:$OLH_HOME/jlib/*
   ```

Oracle Data Integrator Application Adapter for Hadoop Setup

Installation requirements for Oracle Data Integrator (ODI) Application Adapter for Hadoop are provided in these topics:

- System Requirements and Certifications
- Technology-Specific Requirements
- Location of Oracle Data Integrator Application Adapter for Hadoop
- Setting Up the Topology

System Requirements and Certifications

To use Oracle Data Integrator Application Adapter for Hadoop, you must first have Oracle Data Integrator, which is licensed separately from Oracle Big Data Connectors. You can download ODI from the Oracle website at

http://www.oracle.com/technetwork/middleware/data-integrator/downloads/index.html

Oracle Data Integrator Application Adapter for Hadoop requires a minimum version of Oracle Data Integrator 11.1.1.6.0.

Before performing any installation, read the system requirements and certification documentation to ensure that your environment meets the minimum installation requirements for the products that you are installing.

The list of supported platforms and versions is available on Oracle Technology Network:

http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html

Technology-Specific Requirements

The list of supported technologies and versions is available on Oracle Technical Network:

http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html

Location of Oracle Data Integrator Application Adapter for Hadoop

Oracle Data Integrator Application Adapter for Hadoop is available in the xml-reference directory of the Oracle Data Integrator Companion CD.

Setting Up the Topology

To set up the topology, see Chapter 4, "Oracle Data Integrator Application Adapter for Hadoop."

Oracle R Connector for Hadoop Setup

Oracle R Connector for Hadoop requires the installation of a software environment on the Hadoop side and on a client Linux system.

Installing the Software on Hadoop

Oracle Big Data Appliance supports Oracle R Connector for Hadoop without any additional software installation or configuration. However, you do need to verify whether certain R packages are installed. See "Installing Additional R Packages" on page 1-17.

However, to use Oracle R Connector for Hadoop on any other Hadoop cluster, you must create the necessary environment.

Software Requirements for a Third-Party Hadoop Cluster

You must install several software components on a third-party Hadoop cluster to support Oracle R Connector for Hadoop.

Install these components on third-party servers:

- Cloudera's Distribution including Apache Hadoop version 4 (CDH4) or Apache Hadoop 2.0.0, using MapReduce 1.

  Complete the instructions provided by the distributor.

- Hive 0.7.1 or 0.9.0

  See "Installing Hive on a Hadoop Cluster" on page 1-16.

- Sqoop for the execution of functions that connect to Oracle Database. Oracle R Connector for Hadoop does not require Sqoop to install or load.

  See "Installing Sqoop on a Hadoop Cluster" on page 1-15.

- Mahout for the execution of (`orch_lmf_mahout_als.R`).

- Java Virtual Machine (JVM), preferably Java HotSpot Virtual Machine 6.

  Complete the instructions provided at the download site at

  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

- Oracle R Distribution 2.15.1 with all base libraries on all nodes in the Hadoop cluster.

  See "Installing R on a Hadoop Cluster" on page 1-16.

- The ORCH package on each R engine, which must exist on every node of the Hadoop cluster.

  See "Installing the ORCH Package on a Hadoop Cluster" on page 1-17.

- Oracle Loader for Hadoop to support the OLH driver (optional). See "Oracle Loader for Hadoop Setup" on page 1-12.

> **Note:** Do not set `HADOOP_HOME` on the Hadoop cluster. CDH4 does not need it, and it interferes with Oracle R Connector for Hadoop when it checks the status of the JobTracker. This results in the error "Something is terribly wrong with Hadoop MapReduce."
>
> If you must set `HADOOP_HOME` for another application, then also set `HADOOP_LIBEXEC_DIR` in the /etc/bashrc file. For example:
>
> ```
> export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
> ```

Installing Sqoop on a Hadoop Cluster

Sqoop provides a SQL-like interface to Hadoop, which is a Java-based environment. Oracle R Connector for Hadoop uses Sqoop for access to Oracle Database.

> **Note:** Sqoop is required even when using Oracle Loader for Hadoop as a driver for loading data into Oracle Database. Sqoop performs additional functions, such as copying data from a database to HDFS and sending free-form queries to a database. The OLH driver also uses Sqoop to perform operations that Oracle Loader for Hadoop does not support.

To install and configure Sqoop for use with Oracle Database:

1. Install Sqoop if it is not already installed on the server.

For Cloudera's Distribution including Apache Hadoop, see the Sqoop installation instructions in the *CDH Installation Guide* at

http://oracle.cloudera.com/

2. Download the appropriate Java Database Connectivity (JDBC) driver for Oracle Database from Oracle Technology Network at

   http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html

3. Copy the driver JAR file to `$SQOOP_HOME`/lib, which is a directory such as /usr/lib/sqoop/lib.

4. Provide Sqoop with the connection string to Oracle Database.

   ```
   $ sqoop import --connect jdbc_connection_string
   ```

   For example, `sqoop import --connect jdbc:oracle:thin@myhost:1521/orcl`.

Installing Hive on a Hadoop Cluster

Hive provides an alternative storage and retrieval mechanism to HDFS files through a querying language called HiveQL. Oracle R Connector for Hadoop uses the data preparation and analysis features of HiveQL, while enabling you to use R language constructs.

To install Hive:

1. Follow the instructions provided by the distributor (Cloudera or Apache) for installing Hive.

2. Verify that the installation is working correctly:

   ```
   $ hive -H
   usage: hive
    -d,--define <key=value>          Variable subsitution to apply to hive
                                     commands. e.g. -d A=B or --define A=B
       --database <databasename>     Specify the database to use
    -e <quoted-query-string>         SQL from command line
    -f <filename>                    SQL from files
    -H,--help                        Print help information
    -h <hostname>                    connecting to Hive Server on remote host
       --hiveconf <property=value>   Use value for given property
       --hivevar <key=value>         Variable subsitution to apply to hive
                                     commands. e.g. --hivevar A=B
    -i <filename>                    Initialization SQL file
    -p <port>                        connecting to Hive Server on port number
    -S,--silent                      Silent mode in interactive shell
    -v,--verbose                     Verbose mode (echo executed SQL to the
                                     console)
   ```

3. If the command fails or you see warnings in the output, then fix the Hive installation.

Installing R on a Hadoop Cluster

You can download Oracle R Distribution 2.15.1 and get the installation instructions from the website at

http://www.oracle.com/technetwork/indexes/downloads/r-distribution-1532464.html

Alternatively, you can download R from a Comprehensive R Archive Network (CRAN) website at

http://www.r-project.org

Installing the ORCH Package on a Hadoop Cluster

ORCH is the name of the Oracle R Connector for Hadoop package.

To install the ORCH package:

1. Set the environment variables for the supporting software:

```
$ export JAVA_HOME="/usr/lib/jdk6"
$ export R_HOME="/usr/lib64/R"
$ export SQOOP_HOME "/usr/lib/sqoop"
```

2. Unzip the downloaded file:

```
$ unzip orch-version.zip
Archive:  orch-linux-x86_64-2.1.0.zip
 extracting: ORCH2.1.0/ORCH_2.1.0_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.1.0/ORCHcore_2.1.0_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.1.0/OREbase_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.1.0/OREstats_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
```

3. Change to the new directory:

```
$ cd ORCH2.1.0
```

4. Install the packages in the exact order shown here:

```
$ R --vanilla CMD INSTALL OREbase_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
$ R --vanilla CMD INSTALL OREstats_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
$ R --vanilla CMD INSTALL ORCHcore_2.1.0_R_x86_64-unknown-linux-gnu.tar.gz
$ R --vanilla CMD INSTALL ORCH_2.1.0_R_x86_64-unknown-linux-gnu.tar.gz
$ R --vanilla CMD INSTALL ORCHstats_2.1.0_R_x86_64-unknown-linux-gnu.tar.gz
```

Installing Additional R Packages

Your Hadoop cluster must have `libpng-devel` installed on every node. If you are using a cluster running on commodity hardware, then you can follow the same basic procedures. However, you cannot use the `dcli` utility to replicate the commands across all nodes. See the *Oracle Big Data Appliance Owner's Guide* for the syntax of the `dcli` utility.

To install libpng-devel:

1. Log in as `root` to any node in your Hadoop cluster.
2. Check whether `libpng-devel` is already installed:

```
# dcli rpm -qi libpng-devel
bda1node01: package libpng-devel is not installed
bda1node02: package libpng-devel is not installed
     .
     .
     .
```

   If the package is already installed on all servers, then you can skip this procedure.

3. If you need a proxy server to go outside a firewall, then set the `HTTP_PROXY` environment variable:

```
# dcli export HTTP_PROXY="http://proxy.example.com"
```

4. Change to the yum directory:

```
# cd /etc/yum.repos.d
```

5. Download and configure the appropriate configuration file for your version of Linux:

For Enterprise Linux 5 (EL5):

a. Download the yum configuration file:

```
# wget http://public-yum.oracle.com/public-yum-el5.repo
```

b. Open public-yum-el5.repo in a text editor and make these changes:

Under `el5_latest`, set `enabled=1`

Under `el5_addons`, set `enabled=1`

c. Save your changes and exit.

d. Copy the file to the other Oracle Big Data Appliance servers:

```
# dcli -d /etc/yum.repos.d -f public-yum-el5.repo
```

For Oracle Linux 6 (OL6):

a. Download the yum configuration file:

```
# wget http://public-yum.oracle.com/public-yum-ol6.repo
```

b. Open public-yum-ol6.repo in a text editor and make these changes:

Under `ol6_latest`, set `enabled=1`

Under `ol6_addons`, set `enabled=1`

c. Save your changes and exit.

d. Copy the file to the other Oracle Big Data Appliance servers:

```
# dcli -d /etc/yum.repos.d -f public-yum-ol6.repo
```

6. Install the package on all servers:

```
# dcli yum -y install libpng-devel
bda1node01: Loaded plugins: rhnplugin, security
bda1node01: Repository 'bda' is missing name in configuration, using id
bda1node01: This system is not registered with ULN.
bda1node01: ULN support will be disabled.
bda1node01: http://bda1node01-master.us.oracle.com/bda/repodata/repomd.xml:
bda1node01: [Errno 14] HTTP Error 502: notresolvable
bda1node01: Trying other mirror.
     .
     .
     .
bda1node01: Running Transaction
bda1node01: Installing     : libpng-devel                                1/2
bda1node01: Installing     : libpng-devel                                2/2

bda1node01: Installed:
bda1node01: libpng-devel.i386 2:1.2.10-17.el5_8  ibpng-devel.x86_64
2:1.2.10-17.el5_8
```

```
bda1node01: Complete!
bda1node02: Loaded plugins: rhnplugin, security
     .
     .
     .
```

7. Verify that the installation was successful on all servers:

```
# dcli rpm -qi libpng-devel
bda1node01: Name        : libpng-devel                      Relocations: (not
relocatable)
bda1node01: Version     : 1.2.10                                Vendor: Oracle
America
bda1node01: Release     : 17.el5_8                          Build Date: Wed 25
Apr 2012 06:51:15 AM PDT
bda1node01: Install Date: Tue 05 Feb 2013 11:41:14 AM PST    Build Host:
ca-build56.us.oracle.com
bda1node01: Group       : Development/Libraries             Source RPM:
libpng-1.2.10-17.el5_8.src.rpm
bda1node01: Size        : 482483                               License: zlib
bda1node01: Signature   : DSA/SHA1, Wed 25 Apr 2012 06:51:41 AM PDT, Key ID
66ced3de1e5e0159
bda1node01: URL         : http://www.libpng.org/pub/png/
bda1node01: Summary     : Development tools for programs to manipulate PNG
image format files.
bda1node01: Description :
bda1node01: The libpng-devel package contains the header files and static
bda1node01: libraries necessary for developing programs using the PNG (Portable
bda1node01: Network Graphics) library.
     .
     .
     .
```

Providing Remote Client Access to R Users

Whereas R users will run their programs as MapReduce jobs on the Hadoop cluster, they do not typically have individual accounts on that platform. Instead, an external Linux server provides remote access.

Software Requirements for Remote Client Access

To provide access to a Hadoop cluster to R users, install these components on a Linux server:

- The same version of Hadoop as your Hadoop cluster; otherwise, unexpected issues and failures can occur
- The same version of Sqoop as your Hadoop cluster; required only to support copying data in and out of Oracle databases
- Mahout; required only for the `orch.ls` function with the Mahout ALS-WS algorithm
- The same version of the Java Development Kit (JDK) as your Hadoop cluster
- R distribution 2.15.1 with all base libraries
- ORCH R package

To provide access to database objects, you must have the Oracle Advanced Analytics option to Oracle Database. Then you can install this additional component on the Hadoop client:

- Oracle R Enterprise Client Packages

Configuring the Server as a Hadoop Client

You must install Hadoop on the client and minimally configure it for HDFS client use.

To install and configure Hadoop on the client system:

1. Install and configure CDH3 or Apache Hadoop 0.20.2 on the client system. This system can be the host for Oracle Database. If you are using Oracle Big Data Appliance, then complete the procedures for providing remote client access in the *Oracle Big Data Appliance Software User's Guide*. Otherwise, follow the installation instructions provided by the distributor (Cloudera or Apache).
2. Log in to the client system as an R user.
3. Open a Bash shell and enter this Hadoop file system command:

   ```
   $HADOOP_HOME/bin/hadoop fs -ls /user
   ```

4. If you see a list of files, then you are done. If not, then ensure that the Hadoop cluster is up and running. If that does not fix the problem, then you must debug your client Hadoop installation.

Installing Sqoop on a Hadoop Client

Complete the same procedures on the client system for installing and configuring Sqoop as those provided in "Installing Sqoop on a Hadoop Cluster" on page 1-15.

Installing R on a Hadoop Client

You can download R 2.13.2 and get the installation instructions from the Oracle R Distribution website at

`http://oss.oracle.com/ORD/`

When you are done, ensure that users have the necessary permissions to connect to the Linux server and run R.

You may also want to install RStudio Server to facilitate access by R users. See the RStudio website at

`http://rstudio.org/`

Installing the ORCH Package on a Hadoop Client

Complete the procedures on the client system for installing `ORCH` as described in "Installing the Software on Hadoop" on page 1-14.

Installing the Oracle R Enterprise Client Packages (Optional)

To support full access to Oracle Database using R, install the Oracle R Enterprise Release 1.3.1 or later client packages. Without them, Oracle R Connector for Hadoop does not have access to the advanced statistical algorithms provided by Oracle R Enterprise.

> **See Also:** *Oracle R Enterprise User's Guide* for information about installing R and Oracle R Enterprise

2

Oracle SQL Connector for Hadoop Distributed File System

This chapter describes how to use Oracle SQL Connector for Hadoop Distributed File System (HDFS) to facilitate data access between Hadoop and Oracle Database.

This chapter contains the following sections:

About Oracle SQL Connector for HDFS

Using Oracle SQL Connector for HDFS, you can use Oracle Database to access and analyze data residing in Hadoop in these formats:

- Data Pump files in HDFS
- Delimited text files in HDFS
- Hive tables

For other file formats, such as JSON files, you can stage the input in Hive tables before using Oracle SQL Connector for HDFS.

Oracle SQL Connector for HDFS uses external tables to provide Oracle Database with read access to Hive tables, and to delimited text files and Data Pump files in HDFS. An **external table** is an Oracle Database object that identifies the location of data outside of a database. Oracle Database accesses the data by using the metadata provided when the external table was created. By querying the external tables, you can access data stored in HDFS and Hive tables as if that data were stored in tables in an Oracle database.

To create an external table for this purpose, you use the `ExternalTable` command-line tool provided with Oracle SQL Connector for HDFS. You provide `ExternalTable` with information about the data source in Hadoop and about your schema in an Oracle Database. You provide this information either as parameters to the ExternalTable command or in an XML file.

When the external table is ready, you can query the data the same as any other database table. You can query and join data in HDFS or a Hive table with other database-resident data.

You can also perform bulk loads of data into Oracle database tables using SQL. You may prefer that the data resides in an Oracle database—all of it or just a selection—if it is queried routinely.

Getting Started With Oracle SQL Connector for HDFS

The following list identifies the basic steps that you take when using Oracle SQL Connector for HDFS.

1. The first time you use Oracle SQL Connector for HDFS, ensure that the software is installed and configured.

   See "Configuring Your System for Oracle SQL Connector for HDFS" on page 2-5.

2. Log in to the appropriate system, either the Oracle Database system or a node in the Hadoop cluster.

   See "Configuring Your System for Oracle SQL Connector for HDFS" on page 2-5.

3. Create an XML document describing the connections and the data source, unless you are providing these parameters in the `ExternalTable` command.

   See "Describing External Tables" on page 2-16.

4. Create a shell script containing an `ExternalTable` command.

   See "Using the ExternalTable Command-Line Tool" on page 2-6.

5. Run the shell script.

6. If the job fails, then use the diagnostic messages in the output to identify and correct the error. Depending on how far the job progressed before failing, you may need to delete the table definition from the Oracle database before rerunning the script.

7. After the job succeeds, connect to Oracle Database as the owner of the external table. Query the table to ensure that the data is accessible.

8. If the data will be queried frequently, then you may want to load it into a database table. External tables do not have indexes or partitions.

Example 2–1 illustrates these steps.

Example 2–1 Accessing HDFS Data Files from Oracle Database

```
$ cat moviefact_hdfs.sh
# Add environment variables
export OSCH_HOME="/opt/oracle/orahdfs-2.1.0"

hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
       oracle.hadoop.exttab.ExternalTable \
       -conf /home/jdoe/movie/moviefact_hdfs.xml \
       -createTable
```

```
$ cat moviefact_hdfs.xml
<?xml version="1.0"?>
 <configuration>
    <property>
      <name>oracle.hadoop.exttab.tableName</name>
      <value>MOVIE_FACT_EXT_TAB_TXT</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.locationFileCount</name>
      <value>4</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.dataPaths</name>
      <value>/user/jdoe/moviework/data/part*</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.fieldTerminator</name>
      <value>\u0009</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.defaultDirectory</name>
      <value>MOVIE_DIR</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.columnNames</name>
      <value>CUST_ID,MOVIE_ID,GENRE_ID,TIME_ID,RECOMMENDED,ACTIVITY_
ID,RATING,SALES</value>
    </property>
    <property>
      <name>oracle.hadoop.exttab.sourceType</name>
      <value>text</value>
    </property>
    <property>
      <name>oracle.hadoop.connection.url</name>
      <value>jdbc:oracle:thin:@//dbhost:1521/orcl.example.com</value>
    </property>
    <property>
      <name>oracle.hadoop.connection.user</name>
      <value>MOVIEDEMO</value>
    </property>
</configuration>

$ sh moviefact_hdfs.sh
Oracle SQL Connector for HDFS Release 2.1.0 - Production

Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.

Enter Database Password: password]
The create table command succeeded.

CREATE TABLE "MOVIEDEMO"."MOVIE_FACT_EXT_TAB_TXT"
(
  "CUST_ID"                        VARCHAR2(4000),
 "MOVIE_ID"                       VARCHAR2(4000),
 "GENRE_ID"                       VARCHAR2(4000),
 "TIME_ID"                        VARCHAR2(4000),
 "RECOMMENDED"                    VARCHAR2(4000),
 "ACTIVITY_ID"                    VARCHAR2(4000),
 "RATING"                        VARCHAR2(4000),
 "SALES"                          VARCHAR2(4000)
```

```
)
ORGANIZATION EXTERNAL
(
   TYPE ORACLE_LOADER
   DEFAULT DIRECTORY "MOVIE_DIR"
   ACCESS PARAMETERS
   (
     RECORDS DELIMITED BY 0X'0A'
     disable_directory_link_check
     CHARACTERSET AL32UTF8
     STRING SIZES ARE IN CHARACTERS
     PREPROCESSOR "OSCH_BIN_PATH":'hdfs_stream'
     FIELDS TERMINATED BY 0X'2C'
     MISSING FIELD VALUES ARE NULL
     (
       "CUST_ID" CHAR,
       "MOVIE_ID" CHAR,
       "GENRE_ID" CHAR,
       "TIME_ID" CHAR,
       "RECOMMENDED" CHAR,
       "ACTIVITY_ID" CHAR,
       "RATINGS" CHAR,
       "SALES" CHAR
     )
   )
   LOCATION
   (
     'osch-20130314092801-1513-1',
     'osch-20130314092801-1513-2',
     'osch-20130314092801-1513-3',
     'osch-20130314092801-1513-4'
   )
) PARALLEL REJECT LIMIT UNLIMITED;

The following location files were created.

osch-20130314092801-1513-1 contains 1 URI, 12754882 bytes

    12754882 hdfs://bda-ns/user/jdoe/moviework/data/part-00001

osch-20130314092801-1513-2 contains 1 URI, 438 bytes

         438 hdfs://bda-ns/user/jdoe/moviework/data/part-00002

osch-20130314092801-1513-3 contains 1 URI, 432 bytes

         432 hdfs://bda-ns/user/jdoe/moviework/data/part-00003

osch-20130314092801-1513-4 contains 1 URI, 202 bytes

         202 hdfs://bda-ns/user/jdoe/moviework/data/part-00004

SQL*Plus: Release 11.2.0.3.0 Production on Thu Mar 14 14:14:31 2013

Copyright (c) 1982, 2011, Oracle.  All rights reserved.

Enter password: password

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
```

```
With the Partitioning, OLAP and Data Mining options

SQL> SELECT cust_id, movie_id, time_id FROM movie_fact_ext_tab_txt
  2  WHERE rownum < 5;

CUST_ID              MOVIE_ID             TIME_ID
-------------------- -------------------- --------------------
1150211              585                  01-JAN-11
1221463              9870                 01-JAN-11
1002672              1422                 01-JAN-11
1095718              544                  01-JAN-11

SQL> DESCRIBE movie_fact_ext_tab_txt
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                            VARCHAR2(4000)
 MOVIE_ID                                           VARCHAR2(4000)
 GENRE_ID                                           VARCHAR2(4000)
 TIME_ID                                            VARCHAR2(4000)
 RECOMMENDED                                        VARCHAR2(4000)
 ACTIVITY_ID                                        VARCHAR2(4000)
 RATINGS                                            VARCHAR2(4000)
 SALES                                              VARCHAR2(4000)

SQL> CREATE TABLE movie_facts AS
   2 SELECT CAST(cust_id AS VARCHAR2(12)) cust_id,
   3      CAST(movie_id AS VARCHAR2(12)) movie_id,
   4      CAST(genre_id AS VARCHAR(3)) genre_id,
   5      TO_TIMESTAMP(time_id,'YYYY-MM-DD-HH24:MI:SS:FF') time,
   6      TO_NUMBER(recommended) recommended,
   7      TO_NUMBER(activity_id) activity_id,
   8      TO_NUMBER(ratings) ratings,
   9      TO_NUMBER(sales) sales
   10 FROM movie_fact_ext_tab_txt;

SQL> DESCRIBE movie_facts
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                            VARCHAR2(12)
 MOVIE_ID                                           VARCHAR2(12)
 GENRE_ID                                           VARCHAR2(3)
 TIME                                               TIMESTAMP(9)
 RECOMMENDED                                        NUMBER
 ACTIVITY                                           NUMBER
 RATING                                             NUMBER
 SALES                                              NUMBER
```

Configuring Your System for Oracle SQL Connector for HDFS

You can run Oracle SQL Connector for HDFS on either the Oracle Database system or the Hadoop cluster:

- For Hive sources, you must log in to a node in the Hadoop cluster.
- For text and Data Pump format files, you can log in to either the Oracle Database system or a node in the Hadoop cluster.

Oracle SQL Connector for HDFS requires additions to the `HADOOP_CLASSPATH` environment variable on the system where you log in. Your system administrator may have set them up for you when creating your account, or may have left that task for

you. See "Setting Up User Accounts on the Oracle Database System" on page 1-11 and "Setting Up User Accounts on the Hadoop Cluster" on page 1-11.

Setting up the environment variables:

- Verify that `HADOOP_CLASSPATH` includes the path to the JAR files for Oracle SQL Connector for HDFS:

  ```
  path/orahdfs-2.1.0/jlib/*
  ```

- If you are logged in to a Hadoop cluster with Hive data sources, then verify that `HADOOP_CLASSPATH` also includes the Hive JAR files and conf directory. For example:

  ```
  /usr/lib/hive/lib/*
  /etc/hive/conf
  ```

- For your convenience, you can create an `OSCH_HOME` environment variable. The following is the Bash command for setting it on Oracle Big Data Appliance:

  ```
  $ export OSCH_HOME="/opt/oracle/orahdfs-2.1.0"
  ```

> **See Also:** "Oracle SQL Connector for Hadoop Distributed File System Setup" on page 1-4 for instructions for installing the software and setting up user accounts on both systems.
>
> *OSCH_HOME*/doc/README.txt for information about known problems with Oracle SQL Connector for HDFS.

Using the ExternalTable Command-Line Tool

Oracle SQL Connector for HDFS provides a command-line tool named `ExternalTable`. This section describes the basic use of this tool. See "Creating External Tables" on page 2-8 for the command syntax that is specific to your data source format.

About ExternalTable

The `ExternalTable` tool uses the values of several properties to do the following tasks:

- Create an external table
- Populate the location files
- Publish location files to an existing external table
- List the location files
- Describe an external table

You can specify these property values in an XML document or individually on the command line. See "Configuring Oracle SQL Connector for HDFS" on page 2-18..

ExternalTable Command-Line Tool Syntax

This is the full syntax of the `ExternalTable` command-line tool:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-createTable [--noexecute]
  | -publish [--noexecute]
```

```
| -listlocations [--details]
| -getDDL
```

You can either create the `OSCH_HOME` environment variable or replace *OSCH_HOME* in the command syntax with the full path to the installation directory for Oracle SQL Connector for HDFS. On Oracle Big Data Appliance, this directory is:

`/opt/oracle/orahdfs-`*`version`*

For example, you might run the `ExternalTable` command-line tool with a command like this:

```
hadoop jar /opt/oracle/orahdfs-2.1.0/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
      .
      .
      .
```

Parameter Descriptions

-conf *config_file*
Identifies the name of an XML configuration file containing properties needed by the command being executed. See "Configuring Oracle SQL Connector for HDFS" on page 2-18.

-D *property=value*
Assigns a value to a specific property.

-createTable [--noexecute]
Creates an external table definition and publishes the data URIs to the location files of the external table. The output report shows the DDL used to create the external table and lists the contents of the location files.

Use the `--noexecute` option to see the execution plan of the command. The operation is not executed, but the report includes the details of the execution plan and any errors. Oracle recommends that you first execute a `-createTable` command with `--noexecute`.

-publish [--noexecute]
Publishes the data URIs to the location files of an existing external table. Use this option after adding new data files, so that the existing external table can access them.

Use the `--noexecute` option to see the execution plan of the command. The operation is not executed, but the report shows the planned SQL `ALTER TABLE` command and location files. The report also shows any errors. Oracle recommends that you first execute a `-publish` command with `--noexecute`.

See "Publishing the HDFS Data Paths" on page 2-15.

-listLocations [--details]
Shows the location file content as text. With the `--details` option, this command provides a detailed listing. See "What Are Location Files?" on page 2-17.

-getDDL
Prints the table definition of an existing external table. See "Describing External Tables" on page 2-16.

See Also: "Syntax Conventions" on page x

Creating External Tables

You can create external tables automatically using the `ExternalTable` tool provided in Oracle SQL Connector for HDFS.

Creating External Tables with the ExternalTable Tool

To create an external table using the `ExternalTable` tool, follow the instructions for your data source:

- Creating External Tables from Data Pump Format Files
- Creating External Tables from Hive Tables
- Creating External Tables from Delimited Text Files

When the `ExternalTable -createTable` command finishes executing, the external table is ready for use.

To create external tables manually, follow the instructions in "Creating External Tables in SQL" on page 2-15.

ExternalTable Syntax for -createTable

Use the following syntax to create an external table and populate its location files:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-createTable [--noexecute]
```

See Also: "ExternalTable Command-Line Tool Syntax" on page 2-6

Creating External Tables from Data Pump Format Files

Oracle SQL Connector for HDFS supports only Data Pump files produced by Oracle Loader for Hadoop, and does not support generic Data Pump files produced by Oracle Utilities.

Oracle SQL Connector for HDFS creates the external table definition for Data Pump files by using the metadata from the Data Pump file header. It uses the ORACLE_LOADER access driver with the `preprocessor` access parameter. It also uses a special access parameter named `EXTERNAL VARIABLE DATA`, which enables ORACLE_LOADER to read the Data Pump format files generated by Oracle Loader for Hadoop.

Note: Oracle SQL Connector for HDFS requires a patch to Oracle Database 11.2.0.2 before the connector can access Data Pump files produced by Oracle Loader for Hadoop. To download this patch, go to `http://support.oracle.com` and search for bug 14557588.

Release 11.2.0.3 and later releases do not require this patch.

Required Properties

These properties are required:

- `oracle.hadoop.exttab.tableName`
- `oracle.hadoop.exttab.defaultDirectory`
- `oracle.hadoop.exttab.dataPaths`

- oracle.hadoop.exttab.sourceType=datapump
- oracle.hadoop.connection.url
- oracle.hadoop.connection.user

See "Configuring Oracle SQL Connector for HDFS" on page 2-18 for descriptions of the properties used for this data source.

Optional Properties

This property is optional:

- oracle.hadoop.exttab.logDirectory

Defining Properties in XML Files for Data Pump Format Files

Example 2–2 is an XML template containing all the properties that can be used to describe a Data Pump file. To use the template, cut and paste it into a text file, enter the appropriate values to describe your Data Pump file, and delete any optional properties that you do not need. For more information about using XML templates, see "Creating a Configuration File" on page 2-18.

Example 2–2 XML Template with Properties for a Data Pump Format File

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataPaths</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.sourceType</name>
    <value>datapump</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
  </property>

<!-- Optional Properties -->

  <property>
    <name>oracle.hadoop.exttab.logDirectory</name>
    <value>value</value>
  </property>
</configuration>
```

Example

Example 2–3 creates an external table named SALES_DP_XTAB to read Data Pump files.

Example 2–3 Defining an External Table for Data Pump Format Files

Log in as the operating system user that Oracle Database runs under (typically the oracle user), and create a file-system directory:

```
$ mkdir /scratch/sales_dp_dir
```

Create a database directory and grant read and write access to it:

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_dp_dir AS '/scratch/sales_dp_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_dp_dir TO scott;
```

Create the external table:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_DP_XTAB \
-D oracle.hadoop.exttab.sourceType=datapump \
-D oracle.hadoop.exttab.dataPaths=hdfs:///user/scott/olh_sales_dpoutput/ \
-D oracle.hadoop.exttab.defaultDirectory=SALES_DP_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

Creating External Tables from Hive Tables

Oracle SQL Connector for HDFS creates the external table definition from a Hive table by contacting the Hive metastore client to retrieve information about the table columns and the location of the table data. In addition, the Hive table data paths are published to the location files of the Oracle external table.

To read Hive table metadata, Oracle SQL Connector for HDFS requires that the Hive JAR files are included in the HADOOP_CLASSPATH variable. This means that Oracle SQL Connector for HDFS must be installed and running on a computer with a working Hive client.

Ensure that you add the Hive configuration directory to the HADOOP_CLASSPATH environment variable. You must have a correctly functioning Hive client.

For Hive managed tables, the data paths come from the warehouse directory.

For Hive external tables, the data paths from an external location in HDFS are published to the location files of the Oracle external table. Hive external tables can have no data, because Hive does not check if the external location is defined when the table is created. If the Hive table is empty, then one location file is published with just a header and no data URIs.

The Oracle external table is not a "live" Hive table. When changes are made to a Hive table, you must use the ExternalTable tool to either republish the data or create a new external table.

Hive Table Requirements

Oracle SQL Connector for HDFS supports non-partitioned Hive tables that are defined using ROW FORMAT DELIMITED and FILE FORMAT TEXTFILE clauses. Both Hive-managed tables and Hive external tables are supported.

Hive tables can be either bucketed or not bucketed. Table columns with all primitive types from Hive 0.7.1 (CDH3) and the `TIMESTAMP` type are supported.

Required Properties

These properties are required for Hive table sources:

- `oracle.hadoop.exttab.tableName`
- `oracle.hadoop.exttab.defaultDirectory`
- `oracle.hadoop.exttab.sourceType=hive`
- `oracle.hadoop.exttab.hive.tableName`
- `oracle.hadoop.exttab.hive.databaseName`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "Configuring Oracle SQL Connector for HDFS" on page 2-18 for descriptions of the properties used for this data source.

Optional Properties

This property is optional for Hive table sources:

- `oracle.hadoop.exttab.locationFileCount`

Defining Properties in XML Files for Hive Tables

Example 2–4 is an XML template containing all the properties that can be used to describe a Hive table. To use the template, cut and paste it into a text file, enter the appropriate values to describe your Hive table, and delete any optional properties that you do not need. For more information about using XML templates, see "Creating a Configuration File" on page 2-18.

Example 2–4 XML Template with Properties for a Hive Table

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.sourceType</name>
    <value>hive</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.hive.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.hive.databaseName</name>
```

```
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
  </property>

<!-- Optional Properties -->

  <property>
    <name>oracle.hadoop.exttab.locationFileCount</name>
    <value>value</value>
  </property>
</configuration>
```

Example

Example 2–5 creates an external table named `SALES_HIVE_XTAB` to read data from a Hive table. The example defines all the properties on the command line instead of in an XML file.

Example 2–5 Defining an External Table for a Hive Table

Log in as the operating system user that Oracle Database runs under (typically the `oracle` user), and create a file-system directory:

```
$ mkdir /scratch/sales_hive_dir
```

Create a database directory and grant read and write access to it:

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_hive_dir AS '/scratch/sales_hive_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_hive_dir TO scott;
```

Create the external table:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_HIVE_XTAB \
-D oracle.hadoop.exttab.sourceType=hive \
-D oracle.hadoop.exttab.locationFileCount=2 \
-D oracle.hadoop.exttab.hive.tableName=sales_country_us \
-D oracle.hadoop.exttab.hive.databaseName=salesdb \
-D oracle.hadoop.exttab.defaultDirectory=SALES_HIVE_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

Creating External Tables from Delimited Text Files

Oracle SQL Connector for HDFS creates the external table definition for delimited text files using configuration properties that specify the number of columns, the text delimiter, and optionally, the external table column names. All text columns in the external table are VARCHAR2. If column names are not provided, they default to C1 to C*n*, where *n* is the number of columns specified by the `oracle.hadoop.exttab.columnCount` property.

Required Properties

These properties are required for delimited text sources:

- `oracle.hadoop.exttab.tableName`
- `oracle.hadoop.exttab.defaultDirectory`
- `oracle.hadoop.exttab.dataPaths`
- `oracle.hadoop.exttab.columnCount` or `oracle.hadoop.exttab.columnNames`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "Configuring Oracle SQL Connector for HDFS" on page 2-18 for descriptions of the properties used for this data source.

Optional Properties

These properties are optional for delimited text sources:

- `oracle.hadoop.exttab.recordDelimiter`
- `oracle.hadoop.exttab.fieldTerminator`
- `oracle.hadoop.exttab.initialFieldEncloser`
- `oracle.hadoop.exttab.trailingFieldEncloser`
- `oracle.hadoop.exttab.locationFileCount`

Defining Properties in XML Files for Delimited Text Files

Example 2–6 is an XML template containing all the properties that can be used to describe a delimited text file. To use the template, cut and paste it into a text file, enter the appropriate values to describe your data files, and delete any optional properties that you do not need. For more information about using XML templates, see "Creating a Configuration File" on page 2-18.

Example 2–6 XML Template with Properties for a Delimited Text File

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataPaths</name>
    <value>value</value>
  </property>

<!-- Use either columnCount or columnNames -->

  <property>
    <name>oracle.hadoop.exttab.columnCount</name>
```

```
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.columnNames</name>
    <value>value</value>
  </property>

  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
  </property>

<!-- Optional Properties -->

  <property>
    <name>oracle.hadoop.exttab.recordDelimiter</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.fieldTerminator</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.initialFieldEncloser</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.trailingFieldEncloser</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.locationFileCount</name>
    <value>value</value>
  </property>
</configuration>
```

Example

Example 2–7 creates an external table named `SALES_DT_XTAB` from delimited text files.

Example 2–7 Defining an External Table for Delimited Text Files

Log in as the operating system user that Oracle Database runs under (typically the `oracle` user), and create a file-system directory:

```
$ mkdir /scratch/sales_dt_dir
```

Create a database directory and grant read and write access to it:

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_dt_dir AS '/scratch/sales_dt_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_dt_dir TO scott;
```

Create the external table:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_DT_XTAB \
```

```
-D oracle.hadoop.exttab.locationFileCount=2 \
-D oracle.hadoop.exttab.dataPaths="hdfs:///user/scott/olh_sales/*.dat" \
-D oracle.hadoop.exttab.columnCount=10 \
-D oracle.hadoop.exttab.defaultDirectory=SALES_DT_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

Creating External Tables in SQL

You can create an external table manually for Oracle SQL Connector for HDFS. For example, the following procedure enables you to use external table syntax that is not exposed by the `ExternalTable -createTable` command.

Additional syntax might not be supported for Data Pump format files.

To create an external table manually:

1. Use the `-createTable --noexecute` command to generate the external table DDL.
2. Make whatever changes are needed to the DDL.
3. Run the DDL from Step 2 to create the table definition in the Oracle database.
4. Use the `ExternalTable -publish` command to publish the data URIs to the location files of the external table.

Publishing the HDFS Data Paths

The `-createTable` command creates the metadata in Oracle Database and populates the location files with the Universal Resource Identifiers (URIs) of the data files in HDFS. However, you might publish the URIs as a separate step from creating the external table in cases like these:

- You want to publish new data into an already existing external table.
- You created the external table manually instead of using the `ExternalTable` tool.

In both cases, you can use `ExternalTable` with the `-publish` command to populate the external table location files with the URIs of the data files in HDFS. See "Location File Management" on page 2-17.

ExternalTable Syntax for Publish

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-publish [--noexecute]
```

See Also: "ExternalTable Command-Line Tool Syntax" on page 2-6

ExternalTable Command-Line Tool Example

Example 2–8 sets `HADOOP_CLASSPATH` and publishes the HDFS data paths to the external table created in Example 2–3. See "Configuring Your System for Oracle SQL Connector for HDFS" on page 2-5 for more information about setting this environment variable.

Example 2–8 Publishing HDFS Data Paths to an External Table for Data Pump Format Files

This example uses the Bash shell.

```
$ export HADOOP_CLASSPATH="OSCH_HOME/jlib/*"
$ hadoop jar OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_DP_XTAB \
-D oracle.hadoop.exttab.sourceType=datapump \
-D oracle.hadoop.exttab.dataPaths=hdfs:/user/scott/data/ \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.exttab.connection.user=scott -publish
```

In this example:

- `OSCH_HOME` is the full path to the Oracle SQL Connector for HDFS installation directory.
- `SALES_DP_XTAB` is the external table created in Example 2–3.
- `hdfs:/user/scott/data/` is the location of the HDFS data.
- `@myhost:1521/orcl` is the database connection string.

Listing Location File Metadata and Contents

The `-listLocations` command is a debugging and diagnostic utility that enables you to see the location file metadata and contents. You can use this command to verify the integrity of the location files of an Oracle external table.

These properties are required to use this command:

- `oracle.hadoop.exttab.tableName`
- The JDBC connection properties; see "Connection Properties" on page 2-24.

ExternalTable Syntax for -listLocations

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-listLocations [--details]
```

See Also: "ExternalTable Command-Line Tool Syntax" on page 2-6

Describing External Tables

The `-getDDL` command is a debugging and diagnostic utility that prints the definition of an existing external table. This command follows the security model of the PL/SQL `DBMS_METADATA` package, which enables non-privileged users to see the metadata for their own objects.

These properties are required to use this command:

- `oracle.hadoop.exttab.tableName`
- The JDBC connection properties; see "Connection Properties" on page 2-24.

ExternalTable Syntax for -getDDL

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
```

```
[-D property=value]... \
-getDDL
```

See Also: "ExternalTable Command-Line Tool Syntax" on page 2-6

More About External Tables Generated by the ExternalTable Tool

Because external tables are used to access data, all of the features and limitations of external tables apply. Queries are executed in parallel with automatic load balancing. However, update, insert, and delete operations are not allowed and indexes cannot be created on external tables. When an external table is accessed, a full table scan is always performed.

Oracle SQL Connector for HDFS uses the `ORACLE_LOADER` access driver. The hdfs_stream preprocessor script (provided with Oracle SQL Connector for HDFS) modifies the input data to a format that `ORACLE_LOADER` can process.

See Also:

- *Oracle Database Administrator's Guide* for information about external tables
- *Oracle Database Utilities* for more information about external tables, performance hints, and restrictions when you are using the `ORACLE_LOADER` access driver.

What Are Location Files?

A **location file** is a file specified in the location clause of the external table. Oracle SQL Connector for HDFS creates location files that contain only the Universal Resource Identifiers (URIs) of the data files. A **data file** contains the data stored in HDFS.

Enabling Parallel Processing

To enable parallel processing with external tables, you must specify multiple files in the location clause of the external table. The number of files, also known as the **degree of parallelism**, determines the number of child processes started by the external table during a table read. Ideally, the degree of parallelism is no larger than the number of data files, to avoid idle child processes.

Location File Management

The Oracle SQL Connector for HDFS command-line tool, `ExternalTable`, manages the location files of the external table. Location file management involves the following operations:

- Generating new location files in the database directory after checking for name conflicts
- Deleting existing location files in the database directory as necessary
- Publishing data URIs to new location files
- Altering the `LOCATION` clause of the external table to match the new location files

Location file management for the supported data sources is described in the following topics.

Data Pump file format

The `ORACLE_LOADER` access driver is required to access Data Pump files. The driver requires that each location file corresponds to a single Data Pump file in HDFS. Empty location files are not allowed, and so the number of location files in the external table must exactly match the number of data files in HDFS.

Oracle SQL Connector for HDFS automatically takes over location file management and ensures that the number of location files in the external table equals the number of Data Pump files in HDFS.

Delimited files in HDFS and Hive tables

The `ORACLE_LOADER` access driver has no limitation on the number of location files. Each location file can correspond to one or more data files in HDFS. The number of location files for the external table is suggested by the `oracle.hadoop.exttab.locationFileCount` configuration property.

See "Connection Properties" on page 2-24.

Location File Names

This is the format of a location file name:

`osch-`*`timestamp`*`-`*`number`*`-`*`n`*

In this syntax:

- *`timestamp`* has the format *`yyyyMMddhhmmss`*, for example, 20121017103941 for October 17, 2012, at 10:39:41.
- *`number`* is a random number used to prevent location file name conflicts among different tables.
- *`n`* is an index used to prevent name conflicts between location files for the same table.

For example, osch-20121017103941-6807-1.

Configuring Oracle SQL Connector for HDFS

You can pass configuration properties to the `ExternalTable` tool on the command line with the `-D` option, or you can create a configuration file and pass it on the command line with the `-conf` option. These options must precede the command to be executed (`-createTable`, `-publish`, `-listLocations`, or `-getDDL`).

For example, this command uses a configuration file named example.xml:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
       oracle.hadoop.exttab.ExternalTable \
       -conf /home/oracle/example.xml \
       -createTable
```

See "ExternalTable Command-Line Tool Syntax" on page 2-6.

Creating a Configuration File

A configuration file is an XML document with a very simple structure as follows:

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>property</name>
```

```
    <value>value</value>
  </property>
     .
     .
     .
</configuration>
```

Example 2–9 shows a configuration file. See "Configuration Properties" on page 2-19 for descriptions of these properties.

Example 2–9 Configuration File for Oracle SQL Connector for HDFS

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>SH.SALES_EXT_DIR</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataPaths</name>
    <value>/data/s1/*.csv,/data/s2/*.csv</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataCompressionCodec</name>
    <value>org.apache.hadoop.io.compress.DefaultCodec</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>jdbc:oracle:thin:@//myhost:1521/myservicename</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>SH</value>
  </property>
</configuration>
```

Configuration Properties

The following is a complete list of the configuration properties used by the `ExternalTable` command-line tool. The properties are organized into these categories:

- General Properties
- Connection Properties

General Properties

oracle.hadoop.exttab.columnCount
The number of columns for the external table created from delimited text files. The column names are set to C1, C2,... C*n*, where *n* is value of this property.

This property is ignored if `oracle.hadoop.exttab.columnNames` is set.

The `-createTable` command uses this property when `oracle.hadoop.exttab.sourceType=text`.

You must set one of these properties when creating an external table from delimited text files:

- `oracle.hadoop.exttab.columnNames`
- `oracle.hadoop.exttab.columnCount`

oracle.hadoop.exttab.columnNames
A comma-separated list of column names for an external table created from delimited text files. If this property is not set, then the column names are set to C1, C2,... C*n*, where *n* is the value of the `oracle.hadoop.exttab.columnCount` property.

The column names are read as SQL identifiers: unquoted values are capitalized, and double-quoted values stay exactly as entered.

The `-createTable` command uses this property when `oracle.hadoop.exttab.sourceType=text`.

You must set one of these properties when creating an external table from delimited text files:

- `oracle.hadoop.exttab.columnNames`
- `oracle.hadoop.exttab.columnCount`

oracle.hadoop.exttab.dataCompressionCodec
The name of the compression codec class used to decompress the data files. Specify this property when the data files are compressed. Optional.

This property specifies the class name of the compression codec that implements the `org.apache.hadoop.io.compress.CompressionCodec` interface. This codec applies to all data files.

Several standard codecs are available in Hadoop, including the following:

- **bzip2**: `org.apache.hadoop.io.compress.BZip2Codec`
- **gzip**: `org.apache.hadoop.io.compress.GzipCodec`

Default value: None

oracle.hadoop.exttab.dataPaths
A comma-separated list of fully qualified HDFS paths. This parameter enables you to restrict the input by using special pattern-matching characters in the path specification. See Table 2–1. This property is required for the `-createTable` and `-public` commands using Data Pump or delimited text files. The property is ignored for Hive data sources.

For example, to select all files in `/data/s2/`, and only the CSV files in `/data/s7/`, `/data/s8/`, and `/data/s9/`, enter this expression:

```
/data/s2/,/data/s[7-9]/*.csv
```

The external table accesses the data contained in all listed files and all files in listed directories. These files compose a single data set.

The data set can contain compressed files or uncompressed files, but not both.

Table 2–1 Pattern-Matching Characters

Character	Description
?	Matches any single character
*	Matches zero or more characters
[*abc*]	Matches a single character from the character set {*a, b, c*}
[*a-b*]	Matches a single character from the character range {*a...b*}. The character *a* must be less than or equal to *b*.
[^*a*]	Matches a single character that is not from character set or range {*a*}. The carat (^) must immediately follow the left bracket.

Table 2–1 (Cont.) Pattern-Matching Characters

Character	Description
`\c`	Removes any special meaning of character *c*. The backslash is the escape character.
{*ab*\,*cd*}	Matches a string from the string set {*ab*, *cd*}. Precede the comma with an escape character (\) to remove the meaning of the comma as a path separator.
{*ab*\,*c*{*de*\,*fh*}}	Matches a string from the string set {*ab, cde, cfh*}. Precede the comma with an escape character (\) to remove the meaning of the comma as a path separator.

oracle.hadoop.exttab.dataPathFilter
The path filter class. This property is ignored for Hive data sources.

Oracle SQL Connector for HDFS uses a default filter to exclude hidden files, which begin with a dot or an underscore. If you specify another path filter class using the this property, then your filter acts in addition to the default filter. Thus, only visible files accepted by your filter are considered.

oracle.hadoop.exttab.defaultDirectory
Specifies the default directory for the Oracle external table. This directory is used for all input and output files that do not explicitly name a directory object.

Valid value: The name of an existing database directory

Unquoted names are changed to upper case. Double-quoted names are not changed; use them when case-sensitivity is desired. Single-quoted names are not allowed for default directory names.

The `-createTable` command requires this property.

oracle.hadoop.exttab.fieldTerminator
Specifies the field terminator for an external table when `oracle.hadoop.exttab.sourceType=text`. Optional.

Default value: , (comma)

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`. For example, `\t` represents a tab.
- One or more encoded characters in the format `\u`*HHHH*, where *HHHH* is a big-endian hexadecimal representation of the character in UTF-16. For example, `\u0009` represents a tab. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exttab.hive.databaseName
The name of a Hive database that contains the input data table.

The `-createTable` command requires this property when `oracle.hadoop.exttab.sourceType=hive`.

oracle.hadoop.exttab.hive.tableName
The name of an existing Hive table.

The `-createTable` command requires this property when `oracle.hadoop.exttab.sourceType=hive`.

oracle.hadoop.exttab.initialFieldEncloser
Specifies the initial field encloser for an external table created from delimited text files. Optional.

Default value: null; no enclosers are specified for the external table definition.

The `-createTable` command uses this property when `oracle.hadoop.exttab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\u`*HHHH*, where *HHHH* is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exttab.locationFileCount
Specifies the desired number of location files for the external table. Applicable only to non-Data-Pump files.

Default value: 4

This property is ignored if the data files are in Data Pump format. Otherwise, the number of location files is the lesser of:

- The number of data files
- The value of this property

At least one location file is created.

See "Enabling Parallel Processing" on page 2-17 for more information about the number of location files.

oracle.hadoop.exttab.logDirectory
Specifies a database directory where log files, bad files, and discard files are stored. The file names are the default values used by external tables. For example, the name of a log file is the table name followed by _%p.log.

This is an optional property for the `-createTable` command.

These are the default file name extensions:

- Log files: log
- Bad files: bad
- Discard files: dsc

Valid values: An existing Oracle directory object name.

Unquoted names are uppercased. Quoted names are not changed. Table 2–2 provides examples of how values are transformed.

Table 2–2 Examples of Quoted and Unquoted Values

Specified Value	Interpreted Value
`my_log_dir:'sales_tab_%p.log '`	MY_LOG_DIR/sales_tab_%p.log
`'my_log_dir':'sales_tab_%p.log'`	my_log_dir/sales_tab_%p.log
`"my_log_dir":"sales_tab_%p.log"`	my_log_dir/sales_tab_%p.log

oracle.hadoop.exttab.preprocessorDirectory
Specifies the database directory for the preprocessor. The file-system directory must contain the hdfs_stream script.

Default value: `OSCH_BIN_PATH`

The preprocessor directory is used in the `PREPROCESSOR` clause of the external table.

oracle.hadoop.exttab.recordDelimiter
Specifies the record delimiter for an external table created from delimited text files. Optional.

Default value: `\n`

The `-createTable` command uses this parameter when `oracle.hadoop.exttab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\u`*`HHHH`*, where *`HHHH`* is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exttab.sourceType
Specifies the type of source files.

The valid values are `datapump`, `hive`, and `text`.

Default value: `text`

The `-createTable` and `-publish` operations require the value of this parameter.

oracle.hadoop.exttab.tableName
Schema-qualified name of the external table in this format:

schemaName.tableName

If you omit *schemaName*, then the schema name defaults to the connection user name.

Default value: none

Required property for all operations.

oracle.hadoop.exttab.trailingFieldEncloser
Specifies the trailing field encloser for an external table created from delimited text files. Optional.

Default value: null; defaults to the value of `oracle.hadoop.exttab.initialFieldEncloser`

The `-createTable` command uses this property when `oracle.hadoop.exttab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\u`*`HHHH`*, where *`HHHH`* is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

Connection Properties

oracle.hadoop.connection.url
Specifies the database connection string in the thin-style service name format:

```
jdbc:oracle:thin:@//host_name:port/service_name
```

If you are unsure of the service name, then enter this SQL command as a privileged user:

```
SQL> show parameter service
```

If an Oracle wallet is configured as an external password store, then the property value must start with the driver prefix `jdbc:oracle:thin:@` and `db_connect_string` must exactly match the credentials defined in the wallet.

This property takes precedence over all other connection properties.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.user
An Oracle database log-in name. The `externalTable` tool prompts for a password. This parameter is required unless you are using Oracle wallet as an external password store.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.tnsEntryName
Specifies a TNS entry name defined in the tnsnames.ora file.

This property is used with the `oracle.hadoop.connection.tns_admin` property.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.tns_admin
Specifies the directory that contains the tnsnames.ora file. Define this property to use transparent network substrate (TNS) entry names in database connection strings. When using TNSNames with the JDBC thin driver, you must set either this property or the Java `oracle.net.tns_admin` property. When both are set, this property takes precedence over `oracle.net.tns_admin`.

This property must be set when using Oracle Wallet as an external password store. See `oracle.hadoop.connection.wallet_location`.

Default value: The value of the Java `oracle.net.tns_admin` system property

Valid values: A string

oracle.hadoop.connection.wallet_location
A file path to an Oracle wallet directory where the connection credential is stored.

Default value: Not defined

Valid values: A string

When using Oracle Wallet as an external password store, set these properties:

- `oracle.hadoop.connection.wallet_location`

- `oracle.hadoop.connection.url` ***or*** `oracle.hadoop.connection.tnsEntryName`
- `oracle.hadoop.connection.tns_admin`

Performance Tips for Querying Data in HDFS

Parallel processing is extremely important when you are working with large volumes of data. When you use external tables, always enable parallel query with this SQL command:

```
ALTER SESSION ENABLE PARALLEL QUERY;
```

Before loading the data into an Oracle database from the external files created by Oracle SQL Connector for HDFS, enable parallel DDL:

```
ALTER SESSION ENABLE PARALLEL DDL;
```

Before inserting data into an existing database table, enable parallel DML with this SQL command:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Hints such as `APPEND` and `PQ_DISTRIBUTE` also improve performance when you are inserting data.

3

Oracle Loader for Hadoop

This chapter explains how to use Oracle Loader for Hadoop to copy data from Hadoop into tables in an Oracle database. It contains the following sections:

What Is Oracle Loader for Hadoop?

Oracle Loader for Hadoop is an efficient and high-performance loader for fast movement of data from a Hadoop cluster into a table in an Oracle database. It prepartitions the data if necessary and transforms it into a database-ready format. It can also sort records by primary key or user-specified columns before loading the data or creating output files. Oracle Loader for Hadoop uses the parallel processing framework of Hadoop to perform these preprocessing operations, which other loaders typically perform on the database server as part of the load process. Offloading these operations to Hadoop reduces the CPU requirements on the database server, thereby lessening the performance impact on other database tasks.

Oracle Loader for Hadoop is a Java MapReduce application that balances the data across reducers to help maximize performance. It works with a range of input data formats that present the data as records with fields. It can read from sources that have

the data already in a record format (such as Avro files or Hive tables), or it can split the lines of a text file into fields.

You run Oracle Loader for Hadoop using the `hadoop` command-line utility. In the command line, you provide configuration settings with the details of the job. You typically provide these settings in a job configuration file. You can optionally identify the name of a file that maps input fields to the columns of the target table in an Oracle database.

If you have Java programming skills, you can extend the types of data that the loader can handle by defining custom input formats. Then Oracle Loader for Hadoop uses your code to extract the fields and records.

About the Modes of Operation

Oracle Loader for Hadoop operates in two modes:

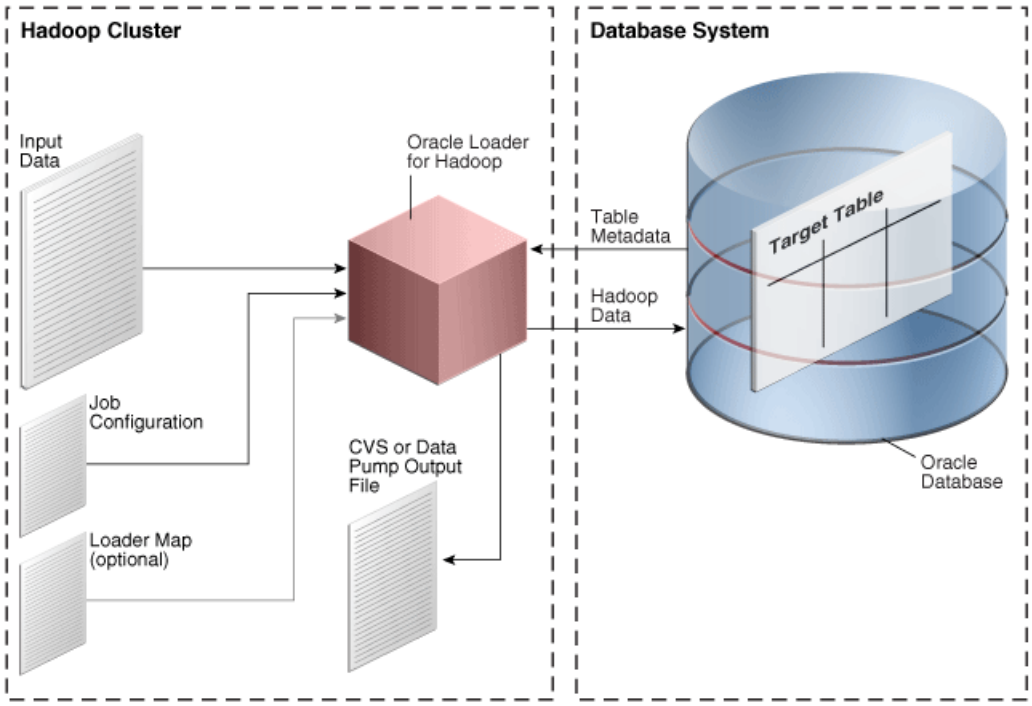
- Online Database Mode
- Offline Database Mode

Online Database Mode

In online database mode, Oracle Loader for Hadoop can connect to the target database using the credentials provided in the job configuration file or in an Oracle wallet. The loader obtains the table metadata from the database. It can insert new records directly into the target table or write them to a file in the Hadoop cluster. You can load records from an output file when the data is needed in the database, or when the database system is less busy.

Figure 3–1 shows the relationships among elements in online database mode.

Figure 3–1 Online Database Mode

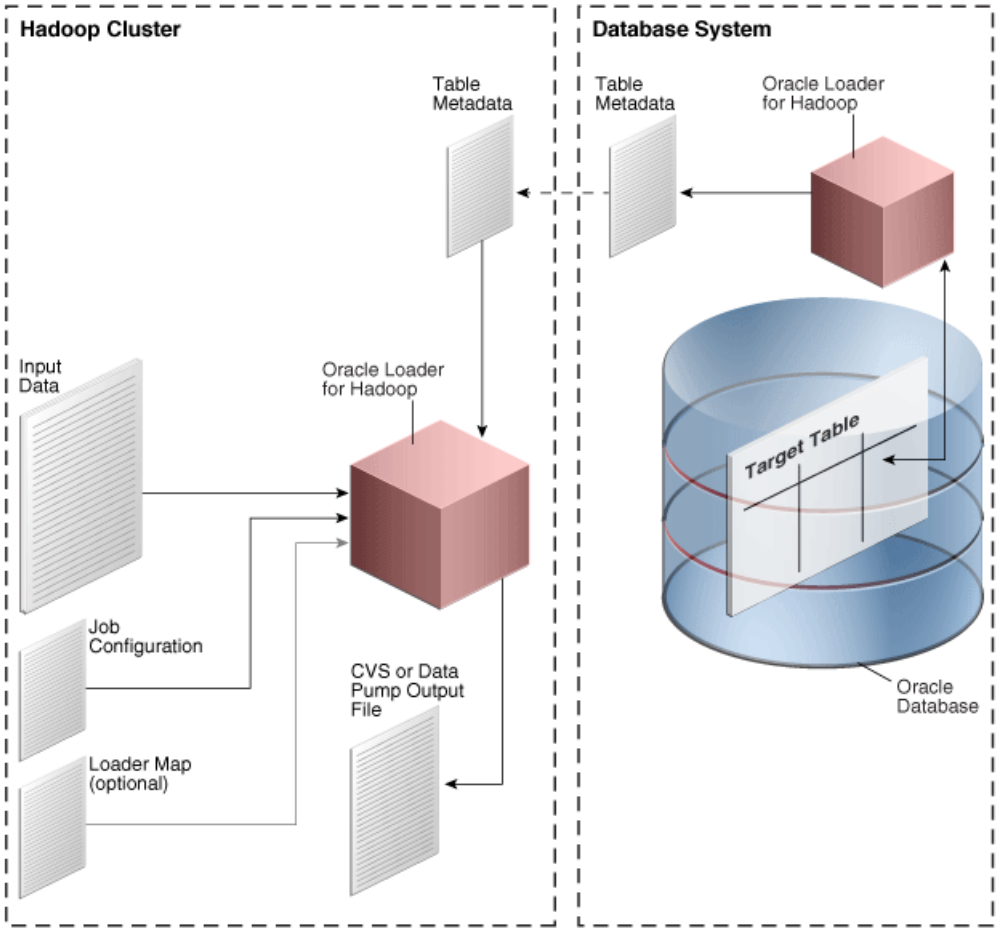


Offline Database Mode

Offline database mode enables you to use Oracle Loader for Hadoop when the Oracle Database system is on a separate network from the Hadoop cluster, or is otherwise inaccessible. In this mode, Oracle Loader for Hadoop uses the information supplied in a table metadata file, which you generate using a separate utility. The loader job stores the output data in binary or text format output files on the Hadoop cluster. Loading the data into Oracle Database is a separate procedure using another utility, such as Oracle SQL Connector for Hadoop Distributed File System (HDFS) or SQL*Loader.

Figure 3–2 shows the relationships among elements in offline database mode. The figure does not show the separate procedure of loading the data into the target table.

Figure 3–2 Offline Database Mode



Getting Started With Oracle Loader for Hadoop

You take the following basic steps when using Oracle Loader for Hadoop:

1. The first time you use Oracle Loader for Hadoop, ensure that the software is installed and configured.

   See "Oracle Loader for Hadoop Setup" on page 1-12.

2. Connect to Oracle Database and create the target table.

   See "Creating the Target Table" on page 3-5.

3. If you are using offline database mode, then generate the table metadata.

   See "Generating the Target Table Metadata for Offline Database Mode" on page 3-8.

4. Log in to either a node in the Hadoop cluster or a system set up as a Hadoop client for the cluster.

5. If you are using offline database mode, then copy the table metadata to the Hadoop system where you are logged in.

6. Create a configuration file. This file is an XML document that describes configuration information, such as access to the target table metadata, the input format of the data, and the output format.

   See "Creating a Job Configuration File" on page 3-6.

7. Create an XML document that maps input fields to columns in the Oracle database table. Optional.

   See "Mapping Input Fields to Target Table Columns" on page 3-14.

8. Create a shell script to run the Oracle Loader for Hadoop job.

   See "Running a Loader Job" on page 3-20.

9. Run the shell script.

10. If the job fails, then use the diagnostic messages in the output to identify and correct the error.

    See "Job Reporting" on page 3-21.

11. After the job succeeds, check the command output for the number of rejected records. If too many records were rejected, then you may need to modify the input format properties.

12. If you generated text files or Data Pump-format files, then load the data into Oracle Database using one of these methods:

    - Create an external table using Oracle SQL Connector for HDFS (online database mode only).

      See Chapter 2.

    - Copy the files to the Oracle Database system and use SQL*Loader or external tables to load the data into the target database table. Oracle Loader for Hadoop generates scripts that you can use for these methods.

      See "About DelimitedTextOutputFormat" on page 3-18 or "About DataPumpOutputFormat" on page 3-19.

13. Connect to Oracle Database as the owner of the target table. Query the table to ensure that the data loaded correctly. If it did not, then modify the input or output format properties as needed to correct the problem.

14. Before running the OraLoader job in a production environment, employ these optimizations:

    - Balancing Loads When Loading Data into Partitioned Tables
    - Optimizing Communications Between Oracle Engineered Systems

Creating the Target Table

Oracle Loader for Hadoop loads data into one **target table**, which must exist in the Oracle database. The table can be empty or contain data already. Oracle Loader for Hadoop does not overwrite existing data.

Create the table the same way that you would create one for any other purpose. It must comply with the following restrictions:

- Supported Data Types for Target Tables
- Supported Partitioning Strategies for Target Tables

Supported Data Types for Target Tables

You can define the target table using any of these data types:

- `BINARY_DOUBLE`
- `BINARY_FLOAT`
- `CHAR`
- `DATE`
- `FLOAT`
- `INTERVAL DAY TO SECOND`
- `INTERVAL YEAR TO MONTH`
- `NCHAR`
- `NUMBER`
- `NVARCHAR2`
- `RAW`
- `TIMESTAMP`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `TIMESTAMP WITH TIME ZONE`
- `VARCHAR2`

The target table can contain columns with unsupported data types, but these columns must be nullable, or otherwise set to a value.

Supported Partitioning Strategies for Target Tables

Partitioning is a database feature for managing and efficiently querying very large tables. It provides a way to decompose a large table into smaller and more manageable pieces called partitions, in a manner entirely transparent to applications.

You can define the target table using any of the following single-level and composite-level partitioning strategies.

- Hash
- Hash-Hash
- Hash-List
- Hash-Range
- Interval

- Interval-Hash
- Interval-List
- Interval-Range
- List
- List-Hash
- List-List
- List-Range
- Range
- Range-Hash
- Range-List
- Range-Range

Oracle Loader for Hadoop does not support reference partitioning or virtual column-based partitioning.

See Also: *Oracle Database VLDB and Partitioning Guide*

Creating a Job Configuration File

A configuration file is an XML document that provides Hadoop with all the information it needs to run a MapReduce job. This file can also provide Oracle Loader for Hadoop with all the information it needs. See "Oracle Loader for Hadoop Configuration Property Reference" on page 3-24.

Configuration properties provide the following information, which is required for all Oracle Loader for Hadoop jobs:

- How to obtain the target table metadata.

  See "About the Target Table Metadata" on page 3-8.

- The format of the input data.

  See "About Input Formats" on page 3-10.

- The format of the output data.

  See "About Output Formats" on page 3-16.

`OraLoader` implements the `org.apache.hadoop.util.Tool` interface and follows the standard Hadoop methods for building MapReduce applications. Thus, you can supply the configuration properties in a file (as shown here) or on the `hadoop` command line. See "Running a Loader Job" on page 3-20.

You can use any text or XML editor to create the file. Example 3–1 provides an example of a job configuration file.

Example 3–1 Job Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

<!--                          Input settings                              -->

 <property>
   <name>mapreduce.inputformat.class</name>
```

```
    <value>oracle.hadoop.loader.lib.input.DelimitedTextInputFormat</value>
  </property>

  <property>
    <name>mapred.input.dir</name>
    <value>/user/oracle/moviedemo/session/*00000</value>
  </property>

  <property>
    <name>oracle.hadoop.loader.input.fieldTerminator</name>
    <value>\u0009</value>
  </property>


<!--                          Output settings                              -->
  <property>
    <name>mapreduce.outputformat.class</name>
    <value>oracle.hadoop.loader.lib.output.OCIOutputFormat</value>
  </property>

  <property>
    <name>mapred.output.dir</name>
    <value>temp_out_session</value>
  </property>

<!--                          Table information                            -->

  <property>
    <name>oracle.hadoop.loader.loaderMapFile</name>
    <value>file:///home/oracle/movie/moviedemo/olh/loaderMap_moviesession.xml
</value>
  </property>

<!--                          Connection information                       -->

<property>
  <name>oracle.hadoop.loader.connection.url</name>
  <value>jdbc:oracle:thin:@${HOST}:${TCPPORT}/${SERVICE_NAME}</value>
</property>

<property>
  <name>TCPPORT</name>
  <value>1521</value>
</property>

<property>
  <name>HOST</name>
  <value>myoraclehost.example.com</value>
</property>

<property>
 <name>SERVICE_NAME</name>
 <value>orcl</value>
</property>

<property>
  <name>oracle.hadoop.loader.connection.user</name>
  <value>MOVIEDEMO</value>
</property>
```

```
<property>
  <name>oracle.hadoop.loader.connection.password</name>
  <value>oracle</value>
  <description> A password in clear text is NOT RECOMMENDED. Use an Oracle wallet
instead.</description>
</property>

</configuration>
```

About the Target Table Metadata

You must provide Oracle Loader for Hadoop with information about the target table. The way that you provide this information depends on whether you run Oracle Loader for Hadoop in online or offline database mode. See "About the Modes of Operation" on page 3-2.

Providing the Connection Details for Online Database Mode

Oracle Loader for Hadoop uses table metadata from the Oracle database to identify the column names, data types, partitions, and so forth. The loader automatically fetches the metadata whenever a JDBC connection can be established.

Oracle recommends that you use a wallet to provide your credentials. To use an Oracle wallet, enter the following properties in the job configuration file:

- `oracle.hadoop.loader.connection.wallet_location`
- `oracle.hadoop.loader.connection.tns_admin`
- `oracle.hadoop.loader.connection.url` ***or*** `oracle.hadoop.loader.connection.tnsEntryName`

Oracle recommends that you do not store passwords in clear text; use an Oracle wallet instead to safeguard your credentials. However, if you are not using an Oracle wallet, then enter these properties:

- `oracle.hadoop.loader.connection.url`
- `oracle.hadoop.loader.connection.user`
- `oracle.hadoop.loader.connection.password`

Generating the Target Table Metadata for Offline Database Mode

Under some circumstances, the loader job cannot access the database, such as when the Hadoop cluster is on a different network than Oracle Database. In such cases, you can use the OraLoaderMetadata utility to extract and store the target table metadata in a file.

To provide target table metadata in offline database mode:

1. Log in to the Oracle Database system.
2. The first time you use offline database mode, ensure that the software is installed and configured on the database system.

   See "Providing Support for Offline Database Mode" on page 1-13.
3. Export the table metadata by running the `OraLoaderMetadata` utility program. See "OraLoaderMetadata Utility" on page 3-9.
4. Copy the generated XML file containing the table metadata to the Hadoop cluster.

5. Use the `oracle.hadoop.loader.tableMetadataFile` property in the job configuration file to specify the location of the XML metadata file on the Hadoop cluster.

   When the loader job runs, it accesses this XML document to discover the target table metadata.

OraLoaderMetadata Utility

Use the following syntax to run the `OraLoaderMetadata` utility on the Oracle Database system. You must enter the `java` command on a single line, although it is shown here on multiple lines for clarity:

```
java oracle.hadoop.loader.metadata.OraLoaderMetadata
   -user userName
   -connection_url connection
   [-schema schemaName]
   -table tableName
   -output fileName.xml
```

To see the OraLoaderMetadata Help file, use the command with no options.

Options

-user *userName*
The Oracle Database user who owns the target table. The utility prompts you for the password.

-connection_url *connection*
The database connection string in the thin-style service name format:

```
jdbc:oracle:thin:@//hostName:port/serviceName
```

If you are unsure of the service name, then enter this SQL command as a privileged user:

```
SQL> show parameter service

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      orcl
```

-schema *schemaName*
The name of the schema containing the target table. Unquoted values are capitalized, and unquoted values are used exactly as entered. If you omit this option, then the utility looks for the target table in the schema specified in the `-user` option.

-table *tableName*
The name of the target table. Unquoted values are capitalized, and unquoted values are used exactly as entered.

-output *fileName*.xml
The output file name used to store the metadata document.

Example 3–2 shows how to store the target table metadata in an XML file.

Example 3–2 Generating Table Metadata

Run the OraLoaderMetadata utility:

```
$ java -cp '/tmp/oraloader-2.1.0-1/jlib/*'
```

```
oracle.hadoop.loader.metadata.OraLoaderMetadata -user HR -connection_url
jdbc:oracle:thin://@localhost:1521/orcl.example.com -table EMPLOYEES -output
employee_metadata.xml
```

The OraLoaderMetadata utility prompts for the database password.

```
Oracle Loader for Hadoop Release 2.1.0 - Production

Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.

[Enter Database Password:] password
```

OraLoaderMetadata creates the XML file in the same directory as the script.

```
$ more employee_metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Oracle Loader for Hadoop Release 2.1.0 - Production

Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.

-->
<DATABASE>
<ROWSET><ROW>
<TABLE_T>
 <VERS_MAJOR>2</VERS_MAJOR>
 <VERS_MINOR>5 </VERS_MINOR>
 <OBJ_NUM>78610</OBJ_NUM>
 <SCHEMA_OBJ>
  <OBJ_NUM>78610</OBJ_NUM>
  <DATAOBJ_NUM>78610</DATAOBJ_NUM>
  <OWNER_NUM>87</OWNER_NUM>
  <OWNER_NAME>HR</OWNER_NAME>
  <NAME>EMPLOYEES</NAME>
     .
     .
     .
```

About Input Formats

An input format reads a specific type of data stored in Hadoop. Several input formats are available, which can read the data formats most commonly found in Hadoop:

- Delimited Text Input Format
- Complex Text Input Formats
- Hive Table Input Format
- Avro Input Format
- Oracle NoSQL Database Input Format

You can also use your own custom input formats. The descriptions of the built-in formats provide information that may help you develop custom Java `InputFormat` classes. See "Custom Input Formats" on page 3-13.

You specify a particular input format for the data that you want to load into a database table, by using the `mapreduce.inputformat.class` configuration property in the job configuration file.

Note: The built-in text formats do not handle header rows or newline characters (`\n`) embedded in quoted values.

Delimited Text Input Format

To load data from a delimited text file, set `mapreduce.inputformat.class` to

`oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`

About DelimitedTextInputFormat

The input file must comply with these requirements:

- Records must be separated by newline characters.
- Fields must be delimited using single-character markers, such as commas or tabs.

Any empty-string token, whether enclosed or unenclosed, is replaced by a null.

`DelimitedTextInputFormat` emulates the tokenization method of SQL*Loader: Terminated by **t**, and optionally enclosed by **ie**, or by **ie** and **te**. `DelimitedTextInputFormat` uses the following syntax rules, where **t** is the field terminator, **ie** is the initial field encloser, **te** is the trailing field encloser, and **c** is one character.

- Line = Token **t** Line | Token**n**
- Token = EnclosedToken | UnenclosedToken
- EnclosedToken = (white-space)* **ie** [(non-**te**)* **te te**]* (non-**te**)* **te** (white-space)*
- UnenclosedToken = (white-space)* (non-**t**)*
- white-space = {**c** | Character.isWhitespace(**c**) and **c**!=**t**}

White space around enclosed tokens (data values) is discarded. For unenclosed tokens, the leading white space is discarded, but not the trailing white space (if any).

This implementation enables you to define custom enclosers and terminator characters, but it hard codes the record terminator as a newline, and white space as Java `Character.isWhitespace`. A white space can be defined as the field terminator, but then that character is removed from the class of white space characters to prevent ambiguity.

Required Configuration Properties

None. The default format separates fields with commas and has no field enclosures.

Optional Configuration Properties

Use one or more of the following properties to define the field delimiters for `DelimitedTextInputFormat`:

- oracle.hadoop.loader.input.fieldTerminator
- oracle.hadoop.loader.input.initialFieldEncloser
- oracle.hadoop.loader.input.trailingFieldEncloser

Use the following property to provide names for the input fields:

- oracle.hadoop.loader.input.fieldNames

Complex Text Input Formats

To load data from text files that are more complex than `DelimitedTextInputFormat` can handle, set `mapreduce.inputformat.class` to

`oracle.hadoop.loader.lib.input.RegexInputFormat`

For example, a web log might delimit one field with quotes and another field with square brackets.

About RegexInputFormat

`RegexInputFormat` requires that records be separated by newline characters. It identifies fields in each text line by matching a regular expression:

- The regular expression must match the entire text line.
- The fields are identified using the capturing groups in the regular expression.

`RegexInputFormat` uses the `java.util.regex` regular expression-based pattern matching engine.

> **See Also:** The *Java Platform Standard Edition 6 Java Reference* for more information about `java.util.regex`:
>
> `http://docs.oracle.com/javase/6/docs/api/java/util/regex/package-summary.html`

Required Configuration Properties

Use the following property to describe the data input file:

- `oracle.hadoop.loader.input.regexPattern`

Optional Configuration Properties

Use the following property to identify the names of all input fields:

- `oracle.hadoop.loader.input.fieldNames`

Use this property to enable case-insensitive matches:

- `oracle.hadoop.loader.input.regexCaseInsensitive`

Hive Table Input Format

To load data from a Hive table, set `mapreduce.inputformat.class` to

`oracle.hadoop.loader.lib.input.HiveToAvroInputFormat`

About HiveToAvroInputFormat

`HiveToAvroInputFormat` imports the entire table, which includes all the files in the Hive table directory or, for partition tables, all files in each partition directory.

Oracle Loader for Hadoop rejects all rows with complex (non-primitive) column values. `UNIONTYPE` fields that resolve to primitive values are supported. See "Handling Rejected Records" on page 3-22.

`HiveToAvroInputFormat` transforms rows in the Hive table into Avro records, and capitalizes the Hive table column names to form the field names. This automatic capitalization improves the likelihood that the field names match the target table column names. See "Mapping Input Fields to Target Table Columns" on page 3-14.

Required Configuration Properties

You must specify the Hive database and table names using the following configuration properties:

- oracle.hadoop.loader.input.hive.databaseName
- oracle.hadoop.loader.input.hive.tableName

Avro Input Format

To load data from binary Avro data files containing standard Avro-format records, set mapreduce.inputformat.class to

oracle.hadoop.loader.lib.input.AvroInputFormat

To process only files with the .avro extension, append *.avro to directories listed in the mapred.input.dir configuration property.

Configuration Properties

None

Oracle NoSQL Database Input Format

To load data from Oracle NoSQL Database, set mapreduce.inputformat.class to

oracle.kv.hadoop.KVAvroInputFormat

This input format is defined in Oracle NoSQL Database 11*g*, Release 2 and later releases.

About KVAvroInputFormat

Oracle Loader for Hadoop uses KVAvroInputFormat to read data directly from Oracle NoSQL Database.

KVAvroInputFormat passes the value but not the key from the key-value pairs in Oracle NoSQL Database. If you must access the Oracle NoSQL Database keys as Avro data values, such as storing them in the target table, then you must create a Java InputFormat class that implements oracle.kv.hadoop.AvroFormatter. Then you can specify the oracle.kv.formatterClass property in the Oracle Loader for Hadoop configuration file.

The KVAvroInputFormat class is a subclass of org.apache.hadoop.mapreduce.InputFormat<oracle.kv.Key, org.apache.avro.generic.IndexedRecord>

See Also: Javadoc for the KVInputFormatBase class at

http://docs.oracle.com/cd/NOSQL/html/index.html

Configuration Properties

None

Custom Input Formats

If the built-in input formats do not meet your needs, then you can write a Java class for a custom input format. The following information describes the framework in which an input format works in Oracle Loader for Hadoop.

About Implementing a Custom Input Format

Oracle Loader for Hadoop gets its input from a class extending `org.apache.hadoop.mapreduce.InputFormat`. You must specify the name of that class in the `mapreduce.inputformat.class` configuration property.

The input format must create `RecordReader` instances that return an Avro `IndexedRecord` input object from the `getCurrentValue` method. Use this method signature:

```
public org.apache.avro.generic.IndexedRecord getCurrentValue()
throws IOException, InterruptedException;
```

Oracle Loader for Hadoop uses the schema of the `IndexedRecord` input object to discover the names of the input fields and map them to the columns of the target table.

About Error Handling

If processing an `IndexedRecord` value results in an error, Oracle Loader for Hadoop uses the object returned by the `getCurrentKey` method of the `RecordReader` to provide feedback. It calls the `toString` method of the key and formats the result in an error message. `InputFormat` developers can assist users in identifying the rejected records by returning one of the following:

- Data file URI
- `InputSplit` information
- Data file name and the record offset in that file

Oracle recommends that you do not return the record in clear text, because it might contain sensitive information; the returned values can appear in Hadoop logs throughout the cluster. See "Logging Rejected Records in Bad Files" on page 3-22.

If a record fails and the key is null, then the loader generates no identifying information.

Supporting Data Sampling

Oracle Loader for Hadoop uses a sampler to improve performance of its MapReduce job. The sampler is multithreaded, and each sampler thread instantiates its own copy of the supplied `InputFormat` class. When implementing a new `InputFormat`, ensure that it is thread-safe. See "Balancing Loads When Loading Data into Partitioned Tables" on page 3-22.

InputFormat Source Code Example

Oracle Loader for Hadoop provides the source code for an `InputFormat` example, which is located in the examples/jsrc/ directory.

The sample format loads data from a simple, comma-separated value (CSV) file. To use this input format, specify `oracle.hadoop.loader.examples.CSVInputFormat` as the value of `mapreduce.inputformat.class` in the job configuration file.

This input format automatically assigns field names of F0, F1, F2, and so forth. It does not have configuration properties.

Mapping Input Fields to Target Table Columns

Mapping identifies which input fields are loaded into which columns of the target table. You may be able to use the automatic mapping facilities, or you can always manually map the input fields to the target columns.

Automatic Mapping

Oracle Loader for Hadoop can automatically map the fields to the appropriate columns when the input data complies with these requirements:

- All columns of the target table are loaded.
- The input data field names in the `IndexedRecord` input object exactly match the column names.
- All input fields that are mapped to `DATE` columns can be parsed using the same Java date format.

Use the `oracle.hadoop.loader.targetTable` configuration property to identify the target table.

Use the `oracle.hadoop.loader.defaultDateFormat` configuration property to specify a default data format that applies to all `DATE` fields.

Manual Mapping

For loads that do not comply with the requirements for automatic mapping, you must create a **loader map**. The loader map enables you to:

- Load data into a subset of the target table columns.
- Create explicit mappings when the input field names are not exactly the same as the database column names.
- Specify different date formats for different input fields.

Use the `oracle.hadoop.loader.loaderMapFile` configuration property to identify the location of the loader map.

Use the `oracle.hadoop.loader.defaultDateFormat` configuration property to specify a default data format that applies to all `DATE` fields (optional).

Creating a Loader Map

To create a loader map, you must know what field names are used by your input format, because you must map them to the appropriate column names.

A loader map has a very simple structure. It can have as many `COLUMN` elements as required by the data:

```
<?xml version="1.0" encoding="UTF-8"?>
<LOADER_MAP>
   <SCHEMA>schemaName</SCHEMA>
   <TABLE>tableName</TABLE>
   <COLUMN field="inputFieldName format="formatString">columnName</COLUMN>
   <COLUMN field="inputFieldName format="formatString">columnName</COLUMN>
   <COLUMN field="inputFieldName format="formatString">columnName</COLUMN>
     .
     .
     .
</LOADER_MAP>
```

Element and Attribute Values:

- *schemaName*: The database schema where the output table resides. Optional.
- *tableName*: The target table.
- *inputFieldName*: The field name of the input data from Hadoop. Required.

- *formatString*: A format string for interpreting the input data. For example, enter a date format field for interpreting dates. Optional.

  You can use this attribute to specify different date formats for different input fields. To specify a default date format that applies to all date input fields, use the `oracle.hadoop.loader.defaultDateFormat` configuration property.

- *columnName*: The name of a column in the target table where the input data will be loaded. Required.

The XML schema definition for the loader map is in `$OLH_HOME/doc/loaderMap.xsd`.

Example Loader Map

Example 3–3 maps the input data to the columns in a table named `EMPLOYEES` in the `HR` schema. Field F3 contains time data.

Example 3–3 Loader Map for SH.EMPLOYEES Target Table

```
<?xml version="1.0" encoding="UTF-8"?>
<LOADER_MAP>
  <SCHEMA>HR</SCHEMA>
  <TABLE>EMPLOYEES</TABLE>
  <COLUMN field="F0">EMPLOYEE_ID</COLUMN>
  <COLUMN field="F1">LAST_NAME</COLUMN>
  <COLUMN field="F2">EMAIL</COLUMN>
  <COLUMN field="F3" format="MM-dd-yyyy">HIRE_DATE</COLUMN>
  <COLUMN field="F4">JOB_ID</COLUMN>
</LOADER_MAP>
```

About Output Formats

In online database mode, you can choose between loading the data directly into an Oracle database table or storing it in a file. In offline database mode, you are restricted to storing the output data in a file, which you can load into the target table as a separate procedure. You specify the output format in the job configuration file using the `mapreduce.outputformat.class` property.

Choose from these output formats:

- JDBC Output Format: Loads the data directly into the target table.
- Oracle OCI Direct Path Output Format: Loads the data directly into the target table.
- Delimited Text Output Format: Stores the data in a local file.
- Oracle Data Pump Output Format: Stores the data in a local file.

JDBC Output Format

You can use a JDBC connection between the Hadoop system and Oracle Database to load the data. The output records of the loader job are loaded directly into the target table by map or reduce tasks as part of the `OraLoader` process, in online database mode. No additional steps are required to load the data.

A JDBC connection must be open between the Hadoop cluster and the Oracle Database system for the duration of the job.

To use this output format, set `mapreduce.outputformat.class` to

`oracle.hadoop.loader.lib.output.JDBCOutputFormat`

About JDBCOutputFormat

`JDBCOutputFormat` uses standard JDBC batching to optimize performance and efficiency. If an error occurs during batch execution, such as a constraint violation, the JDBC driver stops execution immediately. Thus, if there are 100 rows in a batch and the tenth row causes an error, then nine rows are inserted and 91 rows are not.

The JDBC driver does not identify the row that caused the error, and so Oracle Loader for Hadoop does not know the insert status of any of the rows in the batch. It counts all rows in a batch with errors as "in question," that is, the rows may or may not be inserted in the target table. The loader then continues loading the next batch. It generates a load report at the end of the job that details the number of batch errors and the number of rows in question.

One way that you can handle this problem is by defining a unique key in the target table. For example, the HR.EMPLOYEES table has a primary key named EMPLOYEE_ID. After loading the data into HR.EMPLOYEES, you can query it by EMPLOYEE_ID to discover the missing employee IDs.Then you can locate the missing employee IDs in the input data, determine why they failed to load, and try again to load them.

Configuration Properties

To control the batch size, set this property:

`oracle.hadoop.loader.connection.defaultExecuteBatch`

Oracle OCI Direct Path Output Format

You can use the direct path interface of Oracle Call Interface (OCI) to load data into the target table. Each reducer loads into a distinct database partition in online database mode, enabling the performance gains of a parallel load. No additional steps are required to load the data.

The OCI connection must be open between the Hadoop cluster and the Oracle Database system for the duration of the job.

To use this output format, set `mapreduce.outputformat.class` to

`oracle.hadoop.loader.lib.output.OCIOutputFormat`

About OCIOutputFormat

`OCIOutputFormat` has the following restrictions:

- It is available only on the Linux x86.64 platform.
- The MapReduce job must create one or more reducers.
- The target table must be partitioned.
- The target table cannot be a composite interval partitioned table in which the subpartition key contains a `CHAR`, `VARCHAR2`, `NCHAR`, or `NVARCHAR2` column. The load will fail with an error. Oracle Loader for Hadoop does support composite interval partitions in which the subpartition key does not contain a character-type column.

Configuration Properties

To control the size of the direct path stream buffer, set this property:

`oracle.hadoop.loader.output.dirpathBufsize`

Additional Configuration Requirements

If your Hadoop cluster is running the Apache Hadoop distribution or Cloudera's Distribution including Apache Hadoop Version 3 (CDH3), then edit the `$HADOOP_HOME/bin/hadoop` shell script so that it concatenates new values to the existing value. The Apache `hadoop` command begins with an empty `JAVA_LIBRARY_PATH` value and does not import a value from the environment.

If your Hadoop cluster is running CDH4, then the `$HADOOP_HOME/bin/hadoop` command automatically injects this variable value into the Java system property `java.library.path` when the job is created. You do not need to edit the `hadoop` shell script.

Delimited Text Output Format

You can create delimited text output files on the Hadoop cluster. The map or reduce tasks generate delimited text files, using the field delimiters and enclosers that you specify in the job configuration properties. Afterward, you can load the data into an Oracle database as a separate procedure. See "About DelimitedTextOutputFormat" on page 3-18.

This output format can use either an open connection to the Oracle Database system to retrieve the table metadata in online database mode, or a table metadata file generated by the `OraloaderMetadata` utility in offline database mode.

To use this output format, set `mapreduce.outputformat.class` to

`oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat`

About DelimitedTextOutputFormat

Output tasks generate delimited text format files, and one or more corresponding SQL*Loader control files, and SQL scripts for loading with external tables.

If the target table is not partitioned or if `oracle.hadoop.loader.loadByPartition` is `false`, then `DelimitedTextOutputFormat` generates the following files:

- A data file named oraloader-*taskId*-csv-0.dat.
- A SQL*Loader control file named oraloader-csv.ctl for the entire job.
- A SQL script named oraloader-csv.sql to load the delimited text file into the target table.

For partitioned tables, multiple output files are created with the following names:

- Data files: oraloader-*taskId*-csv-*partitionId*.dat
- SQL*Loader control files: oraloader-*taskId*-csv-*partitionId*.ctl
- SQL script: oraloader-csv.sql

In the generated file names, *taskId* is the mapper or reducer identifier, and *partitionId* is the partition identifier.

If the Hadoop cluster is connected to the Oracle Database system, then you can use Oracle SQL Connector for HDFS to load the delimited text data into an Oracle database. See Chapter 2.

Alternatively, you can copy the delimited text files to the database system and load the data into the target table in one of the following ways:

- Use the generated control files to run SQL*Loader and load the data from the delimited text files.

- Use the generated SQL scripts to perform external table loads.

The files are located in the `${map.output.dir}/_olh` directory.

Configuration Properties

The following properties control the formatting of records and fields in the output files:

- `oracle.hadoop.loader.output.escapeEnclosers`
- `oracle.hadoop.loader.output.fieldTerminator`
- `oracle.hadoop.loader.output.initialFieldEncloser`
- `oracle.hadoop.loader.output.trailingFieldEncloser`

Example 3–4 shows a sample SQL*Loader control file that might be generated by an output task.

Example 3–4 Sample SQL*Loader Control File

```
LOAD DATA CHARACTERSET AL32UTF8
INFILE 'oraloader-csv-1-0.dat'
BADFILE 'oraloader-csv-1-0.bad'
DISCARDFILE 'oraloader-csv-1-0.dsc'
INTO TABLE "SCOTT"."CSV_PART" PARTITION(10) APPEND
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
   "ID"      DECIMAL EXTERNAL,
   "NAME"    CHAR,
   "DOB"     DATE 'SYYYY-MM-DD HH24:MI:SS'
)
```

Oracle Data Pump Output Format

You can create Data Pump format files on the Hadoop cluster. The map or reduce tasks generate Data Pump files. Afterward, you can load the data into an Oracle database as a separate procedure. See "About DataPumpOutputFormat" on page 3-19.

This output format can use either an open connection to the Oracle Database system in online database mode, or a table metadata file generated by the `OraloaderMetadata` utility in offline database mode.

To use this output format, set `mapreduce.outputformat.class` to

`oracle.hadoop.loader.lib.output.DataPumpOutputFormat`

About DataPumpOutputFormat

`DataPumpOutputFormat` generates data files with names in this format:

oraloader-*taskId*-dp-*partitionId*.dat

In the generated file names, *taskId* is the mapper or reducer identifier, and *partitionId* is the partition identifier.

If the Hadoop cluster is connected to the Oracle Database system, then you can use Oracle SQL Connector for HDFS to load the Data Pump files into an Oracle database. See Chapter 2.

Alternatively, you can copy the Data Pump files to the database system and load them using a SQL script generated by Oracle Loader for Hadoop. The script performs the following tasks:

1. Creates an external table definition using the `ORACLE_DATAPUMP` access driver. The binary format Oracle Data Pump output files are listed in the `LOCATION` clause of the external table.
2. Creates a directory object that is used by the external table. You must uncomment this command before running the script. To specify the directory name used in the script, set the `oracle.hadoop.loader.extTabDirectoryName` property in the job configuration file.
3. Insert the rows from the external table into the target table. You must uncomment this command before running the script.

The SQL script is located in the `${map.output.dir}/_olh` directory.

See Also:

- *Oracle Database Administrator's Guide* for more information about creating and managing external tables
- *Oracle Database Utilities* for more information about the `ORACLE_DATAPUMP` access driver

Running a Loader Job

To run a job using Oracle Loader for Hadoop, you use the `OraLoader` utility in a `hadoop` command.

The following is the basic syntax:

```
bin/hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf job_config.xml \
-libjars input_file_format1.jar[, input_file_format2.jar...]
```

You can include any generic `hadoop` command-line option. `OraLoader` implements the `org.apache.hadoop.util.Tool` interface and follows the standard Hadoop methods for building MapReduce applications.

Basic Options

-conf *job_config*.xml
Identifies the job configuration file. See "Creating a Job Configuration File" on page 3-6.

-libjars
Identifies the JAR files for the input format.

- When using the example input format, specify $OLH_HOME/jlib/oraloader-examples.jar.
- When using the Hive or Oracle NoSQL Database input formats, you must specify additional JAR files, as described later in this section.
- When using a custom input format, specify its JAR file. (Also remember to add it to `HADOOP_CLASSPATH`.)

Separate multiple file names with commas, and list each one explicitly. Wildcard characters are not allowed.

Oracle Loader for Hadoop prepares internal configuration information for the MapReduce tasks. It stores table metadata information and the dependent Java libraries in the distributed cache, so that they are available to the MapReduce tasks throughout the cluster.

Example of Running OraLoader

The following example uses a built-in input format and a job configuration file named MyConf.xml:

```
HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OLH_HOME/jlib/*"

bin/hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf MyConf.xml -libjars $OLH_HOME/jlib/oraloader-examples.jar
```

See Also:

- For the full `hadoop` command syntax:

  http://hadoop.apache.org/docs/r1.1.1/commands_manual.html#Overview

- For details about the generic options:

  http://hadoop.apache.org/docs/r1.1.1/api/org/apache/hadoop/util/GenericOptionsParser.html

Specifying Hive Input Format JAR Files

When using `HiveToAvroInputFormat`, you must add the Hive configuration directory to the `HADOOP_CLASSPATH` environment variable:

```
HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OLH_HOME/jlib/*:hive_home/lib/*:hive_conf_dir"
```

You must also add the following Hive JAR files, in a comma-separated list, to the `-libjars` option of the `hadoop` command. Replace the stars (*) with the complete file names on your system:

- hive-exec-*.jar
- hive-metastore-*.jar
- libfb303*.jar

This example shows the full file names in Cloudera's Distribution including Apache Hadoop (CDH) 4.1.2:

```
# bin/hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf MyConf.xml \
-libjars hive-exec-0.9.0-cdh4.1.2.jar,hive-metastore-0.9.0-cdh4.1.2.jar, \
libfb303-0.7.0.jar \
```

Specifying Oracle NoSQL Database Input Format JAR Files

When using `KVAvroInputFormat` from Oracle NoSQL Database 11*g*, Release 2, you must include `$KVHOME/lib/kvstore.jar` in your `HADOOP_CLASSPATH` and you must include the `-libjars` option in the `hadoop` command:

```
bin/hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf MyConf.xml \
-libjars $KVHOME/lib/kvstore.jar
```

Job Reporting

Oracle Loader for Hadoop consolidates reporting information from individual tasks into a file named `${map.output.dir}/_olh/oraloader-report.txt`. Among other

statistics, the report shows the number of errors, broken out by type and task, for each mapper and reducer.

Handling Rejected Records

Oracle Loader for Hadoop may reject input records for a variety of reasons, such as:

- Errors in the loader map file
- Missing fields in the input data
- Records mapped to invalid table partitions
- Badly formed records, such as dates that do not match date format or records that do not match regular expression patterns

Logging Rejected Records in Bad Files

By default, Oracle Loader for Hadoop does not log the rejected records into Hadoop logs; it only logs information on how to identify the rejected records. This practice prevents user-sensitive information from being stored in Hadoop logs across the cluster.

You can direct Oracle Loader for Hadoop to log rejected records by setting the `oracle.hadoop.loader.logBadRecords` configuration property to `true`. Then Oracle Loader for Hadoop logs bad records into one or more "bad" files in the `_olh/` directory under the job output directory.

Setting a Job Reject Limit

Problems like using the wrong loader map file can cause Oracle Loader for Hadoop to reject every record in the input. To mitigate the loss of time and resources, Oracle Loader for Hadoop aborts the job after rejecting 1000 records.

You can change the maximum number of rejected records allowed by setting the `oracle.hadoop.loader.rejectLimit` configuration property. A negative value turns off the reject limit and allows the job to run to completion regardless of the number of rejected records.

Balancing Loads When Loading Data into Partitioned Tables

The goal of load balancing is to generate a MapReduce partitioning scheme that assigns approximately the same amount of work to all reducers.

The sampling feature of Oracle Loader for Hadoop balances loads across reducers when data is loaded into a partitioned database table. It generates an efficient MapReduce partitioning scheme that assigns database partitions to the reducers.

The execution time of a reducer is usually proportional to the number of records that it processes—the more records, the longer the execution time. When sampling is disabled, all records from a given database partition are sent to one reducer. This can result in unbalanced reducer loads, because some database partitions may have more records than others. Because the execution time of a Hadoop job is usually dominated by the execution time of its slowest reducer, unbalanced reducer loads slow down the entire job.

Using the Sampling Feature

You can turn the sampling feature on or off by setting the `oracle.hadoop.loader.sampler.enableSampling` configuration property. Sampling is turned on by default.

Tuning Load Balancing

These job configuration properties control the quality of load balancing:

- `oracle.hadoop.loader.sampler.maxLoadFactor`
- `oracle.hadoop.loader.sampler.loadCI`

The sampler uses the expected reducer load factor to evaluate the quality of its partitioning scheme. The **load factor** is the relative overload for each reducer, calculated as (*assigned_load* - *ideal_load*)/*ideal_load*. This metric indicates how much a reducer's load deviates from a perfectly balanced reducer load. A load factor of 1.0 indicates a perfectly balanced load (no overload).

Small load factors indicate better load balancing. The `maxLoadFactor` default of 0.05 means that no reducer is ever overloaded by more than 5%. The sampler guarantees this `maxLoadFactor` with a statistical confidence level determined by the value of `loadCI`. The default value of `loadCI` is 0.95, which means that any reducer's load factor exceeds `maxLoadFactor` in only 5% of the cases.

There is a trade-off between the execution time of the sampler and the quality of load balancing. Lower values of `maxLoadFactor` and higher values of `loadCI` achieve more balanced reducer loads at the expense of longer sampling times. The default values of `maxLoadFactor=0.05` and `loadCI=0.95` are a good trade-off between load balancing quality and execution time.

Tuning Sampling Behavior

By default, the sampler runs until it collects just enough samples to generate a partitioning scheme that satisfies the `maxLoadFactor` and `loadCI` criteria.

However, you can limit the sampler running time by setting the `oracle.hadoop.loader.sampler.maxSamplesPct` property, which specifies the maximum number of sampled records.

When Does Oracle Loader for Hadoop Use the Sampler's Partitioning Scheme?

Oracle Loader for Hadoop uses the generated partitioning scheme only if sampling is successful. A sampling is successful if it generates a partitioning scheme with a maximum reducer load factor of (1+ `maxLoadFactor`) guaranteed at a statistical confidence level of `loadCI`.

The default values of `maxLoadFactor`, `loadCI`, and `maxSamplesPct` allow the sampler to successfully generate high-quality partitioning schemes for a variety of different input data distributions. However, the sampler might be unsuccessful in generating a partitioning scheme using custom property values, such as when the constraints are too rigid or the number of required samples exceeds the user-specified maximum of `maxSamplesPct`. In these cases, Oracle Loader for Hadoop generates a log message identifying the problem, partitions the records using the database partitioning scheme, and does not guarantee load balancing.

Alternatively, you can reset the configuration properties to less rigid values. Either increase `maxSamplesPct`, or decrease `maxLoadFactor` or `loadCI`, or both.

Resolving Memory Issues

A custom input format may return input splits that do not fit in memory. If this happens, the sampler returns an out-of-memory error on the client node where the loader job is submitted.

To resolve this problem:

- Increase the heap size of the JVM where the job is submitted.
- Adjust the following properties:
  - oracle.hadoop.loader.sampler.hintMaxSplitSize
  - oracle.hadoop.loader.sampler.hintNumMapTasks

If you are developing a custom input format, then see "Custom Input Formats" on page 3-13.

What Happens When a Sampling Feature Property Has an Invalid Value?

If any configuration properties of the sampling feature are set to values outside the accepted range, an exception is not returned. Instead, the sampler prints a warning message, resets the property to its default value, and continues executing.

Optimizing Communications Between Oracle Engineered Systems

If you are using Oracle Loader for Hadoop to load data from Oracle Big Data Appliance to Oracle Exadata Database Machine, then you can increase throughput by configuring the systems to use Sockets Direct Protocol (SDP) over the InfiniBand private network. This setup provides an additional connection attribute whose sole purpose is serving connections to Oracle Database to load data.

To specify SDP protocol:

1. Add JVM options to the `HADOOP_OPTS` environment variable to enable JDBC SDP export:

   ```
   HADOOP_OPTS="-Doracle.net.SDP=true -Djava.net.preferIPv4Stack=true"
   ```

2. Configure the Oracle listener on Exadata with a specific port address for SDP (such as 1522). In the job configuration file, set `oracle.hadoop.loader.connection.url` to SDP using this syntax:

   ```
   jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=SDP)
      (HOST=hostName) (PORT=portNumber))
      (CONNECT_DATA=(SERVICE_NAME=serviceName)))
   ```

   Replace *hostName*, *portNumber*, and *serviceName* with the appropriate values to identify Oracle Database on your Oracle Exadata Database Machine.

   See Also: *Oracle Big Data Appliance Software User's Guide* for more information about configuring communications over InfiniBand

Oracle Loader for Hadoop Configuration Property Reference

`OraLoader` uses the standard methods of specifying configuration properties in the hadoop command. You can use the `-conf` option to identify configuration files, and the `-D` option to specify individual properties. See "Running a Loader Job" on page 3-20.

This section describes the OraLoader configuration properties and a few generic Hadoop MapReduce properties that you typically must set for an `OraLoader` job. A configuration file showing all OraLoader properties is in $OLH_HOME/doc/oraloader-conf.xml.

> **See Also:** Hadoop documentation for job configuration files at
>
> `http://wiki.apache.org/hadoop/JobConfFile`

MapReduce Configuration Properties

mapred.job.name
Type: String

Default Value: `OraLoader`

Description: The Hadoop job name. A unique name can help you monitor the job using tools such as the Hadoop JobTracker web interface and Cloudera Manager.

mapred.input.dir
Type: String

Default Value: Not defined

Description: A comma-separated list of input directories.

mapreduce.inputformat.class
Type: String

Default Value: Not defined

Description: Identifies the format of the input data. You can enter one of the following built-in input formats, or the name of a custom `InputFormat` class:

- `oracle.hadoop.loader.lib.input.AvroInputFormat`
- `oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`
- `oracle.hadoop.loader.lib.input.HiveToAvroInputFormat`
- `oracle.hadoop.loader.lib.input.RegexInputFormat`
- `oracle.kv.hadoop.KVAvroInputFormat`

See "About Input Formats" on page 3-10 for descriptions of the built-in input formats.

mapred.output.dir
Type: String

Default Value: Not defined

Description: A comma-separated list of output directories, which cannot exist before the job runs. Required.

mapreduce.outputformat.class
Type: String

Default Value: Not defined

Description: Identifies the output type. The values can be:

- `oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat`

  Writes data records to delimited text format files such as comma-separated values (CSV) format files.

- `oracle.hadoop.loader.lib.output.JDBCOutputFormat`

  Inserts data records into the target table using JDBC.

- `oracle.hadoop.loader.lib.output.OCIOutputFormat`

  Inserts rows into the target table using the Oracle OCI Direct Path interface.

- `oracle.hadoop.loader.lib.output.DataPumpOutputFormat`

  Writes rows into binary format files that can be loaded into the target table using an external table.

See "About Output Formats" on page 3-16.

OraLoader Configuration Properties

oracle.hadoop.loader.badRecordFlushInterval

Type: Integer

Default Value: `500`

Description: Sets the maximum number of records that a task attempt can log before flushing the log file. This setting limits the number of records that can be lost when the record reject limit (`oracle.hadoop.loader.rejectLimit`) is reached and the job stops running.

The `oracle.hadoop.loader.logBadRecords` property must be set to `true` for a flush interval to take effect.

oracle.hadoop.loader.compressionFactors

Type: Decimal

Default Value: `BASIC=5.0,OLTP=5.0,QUERY_LOW=10.0,QUERY_HIGH=10.0,ARCHIVE_LOW=10.0,ARCHIVE_HIGH=10.0`

Description: Defines the compression factor for different types of compression. The value is a comma-delimited list of *name*=*value* pairs. The name can be one of the following keywords:

```
ARCHIVE_HIGH
ARCHIVE_LOW
BASIC
OLTP
QUERY_HIGH
QUERY_LOW
```

oracle.hadoop.loader.connection.defaultExecuteBatch

Type: Integer

Default Value: `100`

Description: The number of records inserted in one trip to the database. It applies only to `JDBCOutputFormat` and `OCIOutputFormat`.

Specify a value greater than or equal to 1. Although the maximum value is unlimited, very large batch sizes are not recommended because they result in a large memory footprint without much increase in performance.

A value less than 1 sets the property to the default value.

oracle.hadoop.loader.connection.oci_url

Type: String

Default Value: Value of `oracle.hadoop.loader.connection.url`

Description: The database connection string used by `OCIOutputFormat`. This property enables the OCI client to connect to the database using different connection parameters than the JDBC connection URL.

The following example specifies Socket Direct Protocol (SDP) for OCI connections.

```
(DESCRIPTION=(ADDRESS_LIST=
(ADDRESS=(PROTOCOL=SDP)(HOST=myhost)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))
```

This connection string does not require a "jdbc:oracle:thin:@" prefix. All characters up to and including the first at-sign (@) are removed.

oracle.hadoop.loader.connection.password

Type: String

Default Value: Not defined

Description: Password for the connecting user. Oracle recommends that you do not store your password in clear text. Use an Oracle wallet instead.

oracle.hadoop.loader.connection.sessionTimeZone

Type: String

Default Value: `LOCAL`

Description: Alters the session time zone for database connections. Valid values are:

- [+|-]*hh*:*mm*: Hours and minutes before or after Coordinated Universal Time (UTC), such as `-5:00` for Eastern Standard Time
- `LOCAL`: The default time zone of the JVM
- *time_zone_region*: A valid JVM time zone region, such as EST (for Eastern Standard Time) or America/New_York

This property also determines the default time zone for input data that is loaded into the following database column types: `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE`.

oracle.hadoop.loader.connection.tns_admin

Type: String

Default Value: Not defined

Description: File path to a directory on each node of the Hadoop cluster, which contains SQL*Net configuration files such as `sqlnet.ora` and `tnsnames.ora`. Set this property so that you can use TNS entry names in database connection strings.

You must set this property when using an Oracle wallet as an external password store. See `oracle.hadoop.loader.connection.wallet_location`.

oracle.hadoop.loader.connection.tnsEntryName

Type: String

Default Value: Not defined

Description: A TNS entry name defined in the `tnsnames.ora` file. Use this property with `oracle.hadoop.loader.connection.tns_admin`.

oracle.hadoop.loader.connection.url

Type: String

Default Value: Not defined

Description: The URL of the database connection. This property overrides all other connection properties.

If an Oracle wallet is configured as an external password store, then the property value must start with the `jdbc:oracle:thin:@` driver prefix, and the database connection string must exactly match the credential in the wallet. See `oracle.hadoop.loader.connection.wallet_location`.

The following examples show valid values of connection URLs:

- Oracle Net Format:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
         (ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)))
         (CONNECT_DATA=(SERVICE_NAME=example_service_name)))
```

- Oracle Net Format for InfiniBand:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=SDP)
         (HOST=myhost) (PORT=1522))
         (CONNECT_DATA=(SERVICE_NAME=example_service_name)))
```

- TNS Entry Format:

```
jdbc:oracle:thin:@myTNSEntryName
```

- Thin Style:

```
jdbc:oracle:thin:@//myhost:1521/my_db_service_name

jdbc:oracle:thin:user/password@//myhost:1521/my_db_service_name
```

oracle.hadoop.loader.connection.user
Type: String

Default Value: Not defined

Description: A database user name. When using online database mode, you must set either this property or `oracle.hadoop.loader.connection.wallet_location`.

oracle.hadoop.loader.connection.wallet_location
Type: String

Default Value: Not defined

Description: File path to an Oracle wallet directory on each node of the Hadoop cluster, where the connection credentials are stored.

When using an Oracle wallet, you must also set the following properties:

- `oracle.hadoop.loader.connection.tns_admin`
- `oracle.hadoop.loader.connection.url` ***or*** `oracle.hadoop.loader.connection.tnsEntryName`

oracle.hadoop.loader.defaultDateFormat
Type: String

Default Value: `yyyy-MM-dd HH:mm:ss`

Description: Parses an input field into a `DATE` column using a `java.text.SimpleDateformat` pattern and the default locale. If the input file requires different patterns for different fields, then use a loader map file. See "Manual Mapping" on page 3-15.

oracle.hadoop.loader.enableSorting
Type: Boolean

Default Value: `true`

Description: Controls whether output records within each reducer group are sorted. Use the `oracle.hadoop.loader.sortKey` property to identify the columns of the target table to sort by. Otherwise, Oracle Loader for Hadoop sorts the records by the primary key.

oracle.hadoop.loader.extTabDirectoryName
Type: String

Default Value: `OLH_EXTTAB_DIR`

Description: The name of the database directory object for the external table `LOCATION` data files. Oracle Loader for Hadoop does not copy data files into this directory; the file output formats generate a SQL file containing external table DDL, where the directory name appears.

This property applies only to `DelimitedTextOutputFormat` and `DataPumpOutputFormat`.

oracle.hadoop.loader.input.fieldNames
Type: String

Default Value: `F0,F1,F2,...`

Description: A comma-delimited list of names for the input fields.

For the built-in input formats, specify names for all fields in the data, not just the fields of interest. If an input line has more fields than this property has field names, then the extra fields are discarded. If a line has fewer fields than this property has field names, then the extra fields are set to null. See "Mapping Input Fields to Target Table Columns" on page 3-14 for loading only selected fields.

The names are used to create the Avro schema for the record, so they must be valid JSON name strings.

oracle.hadoop.loader.input.fieldTerminator
Type: String

Default Value: , (comma)

Description: A character that indicates the end of an input field for `DelimitedTextInputFormat`. The value can be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding.

oracle.hadoop.loader.input.hive.databaseName
Type: String

Default Value: Not defined

Description: The name of the Hive database where the input table is stored.

oracle.hadoop.loader.input.hive.tableName
Type: String

Default Value: Not defined

Description: The name of the Hive table where the input data is stored.

oracle.hadoop.loader.input.initialFieldEncloser
Type: String

Default Value: Not defined

Description: A character that indicates the beginning of a field. The value can be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding. To restore the default setting (no encloser), enter a zero-length value. A field encloser cannot equal the terminator or white-space character defined for the input format.

When this property is set, the parser attempts to read each field as an enclosed token (value) before reading it as an unenclosed token. If the field enclosers are not set, then the parser reads each field as an unenclosed token.

If you set this property but not `oracle.hadoop.loader.input.trailingFieldEncloser`, then the same value is used for both properties.

oracle.hadoop.loader.input.regexCaseInsensitive

Type: Boolean

Default Value: `false`

Description: Controls whether pattern matching is case-sensitive. Set to `true` to ignore case, so that "string" matches "String", "STRING", "string", "StRiNg", and so forth. By default, "string" matches only "string".

This property is the same as the`input.regex.case.insensitive` property of `org.apache.hadoop.hive.contrib.serde2.RegexSerDe`.

oracle.hadoop.loader.input.regexPattern

Type: Text

Default Value: Not defined

Description: The pattern string for a regular expression.

The regular expression must match each text line in its entirety. For example, a correct regex pattern for input line `"a,b,c,"` is `"([^,]*),([^,]*),([^,]*),"`. However, `"([^,]*),"` is invalid, because the expression is *not* applied repeatedly to a line of input text.

`RegexInputFormat` uses the capturing groups of regular expression matching as fields. The special group zero is ignored because it stands for the entire input line.

This property is the same as the `input.regex` property of `org.apache.hadoop.hive.contrib.serde2.RegexSerDe`.

> **See Also:** For descriptions of regular expressions and capturing groups, the entry for `java.util.regex` in the *Java Platform Standard Edition 6 API Specification* at
>
> `http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html`

oracle.hadoop.loader.input.trailingFieldEncloser

Type: String

Default Value: The value of `oracle.hadoop.loader.input.initialFieldEncloser`

Description: Identifies a character that marks the end of a field. The value can be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding. For no trailing encloser, enter a zero-length value.

A field encloser cannot be the terminator or a white-space character defined for the input format.

If the trailing field encloser character is embedded in an input field, then the character must be doubled up to be parsed as literal text. For example, an input field must have `''` (two single quotes) to load `'` (one single quote).

If you set this property, then you must also set `oracle.hadoop.loader.input.initialFieldEncloser`.

oracle.hadoop.loader.libjars

Type: String

Default Value:

```
${oracle.hadoop.loader.olh_home}/jlib/ojdbc6.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-utility.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-mapping.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-collation.jar,
${oracle.hadoop.loader.olh_home}/jlib/oraclepki.jar,
${oracle.hadoop.loader.olh_home}/jlib/osdt_cert.jar,
${oracle.hadoop.loader.olh_home}/jlib/osdt_core.jar,
${oracle.hadoop.loader.olh_home}/jlib/commons-math-2.2.jar,
${oracle.hadoop.loader.olh_home}/jlib/jackson-core-asl-1.8.8.jar,
${oracle.hadoop.loader.olh_home}/jlib/jackson-mapper-asl-1.8.8.jar,
${oracle.hadoop.loader.olh_home}/jlib/avro-1.6.3.jar,
${oracle.hadoop.loader.olh_home}/jlib/avro-mapred-1.6.3.jar
```

Description: A comma-delimited list of JAR files, which is appended to the value of the `hadoop -libjars` command-line argument. You can add your application JAR files to the `CLASSPATH` by using this property either instead of, or with, the `-libjars` option to the `hadoop` command.

A leading comma or consecutive commas are invalid.

oracle.hadoop.loader.loadByPartition

Type: Boolean

Default Value: `true`

Description: Specifies a partition-aware load. Oracle Loader for Hadoop organizes the output by partition for all output formats on the Hadoop cluster; this task does not impact the resources of the database system.

`DelimitedTextOutputFormat` and `DataPumpOutputFormat` generate multiple files, and each file contains the records from one partition. For `DelimitedTextOutputFormat`, this property also controls whether the `PARTITION` keyword appears in the generated control files for SQL*Loader.

`OCIOutputFormat` requires partitioned tables. If you set this property to false, then `OCIOutputFormat` turns it back on. For the other output formats, you can set `loadByPartition` to false, so that Oracle Loader for Hadoop handles a partitioned table as if it were unpartitioned.

oracle.hadoop.loader.loaderMapFile

Type: String

Default Value: Not defined

Description: Path to the loader map file. Use the `file://` syntax to specify a local file, for example:

```
file:///home/jdoe/loadermap.xml
```

oracle.hadoop.loader.logBadRecords
Type: Boolean

Default Value: `false`

Description: Controls whether Oracle Loader for Hadoop logs bad records to a file.

This property applies only to records rejected by input formats and mappers. It does not apply to errors encountered by the output formats or by the sampling feature.

oracle.hadoop.loader.log4j.propertyPrefix
Type: String

Default Value: `log4j.logger.oracle.hadoop.loader`

Description: Identifies the prefix used in Apache log4j properties loaded from its configuration file.

Oracle Loader for Hadoop enables you to specify log4j properties in the `hadoop` command using the `-conf` and `-D` options. For example:

```
-D log4j.logger.oracle.hadoop.loader.OraLoader=DEBUG
-D log4j.logger.oracle.hadoop.loader.metadata=INFO
```

All configuration properties starting with this prefix are loaded into log4j. They override the settings for the same properties that log4j loaded from ${log4j.configuration}. The overrides apply to the Oracle Loader for Hadoop job driver, and its map and reduce tasks.

The configuration properties are copied to log4j with RAW values; any variable expansion is done in the context of log4j. Any configuration variables to be used in the expansion must also start with this prefix.

oracle.hadoop.loader.olh_home
Type: String

Default Value: Value of the `OLH_HOME` environment variable

Description: The path of the Oracle Loader for Hadoop home directory on the node where you start the OraLoader job. This path identifies the location of the required libraries.

oracle.hadoop.loader.olhcachePath
Type: String

Default Value: `${mapred.output.dir}/.../olhcache`

Description: Identifies the full path to an HDFS directory where Oracle Loader for Hadoop can create files that are loaded into the MapReduce distributed cache.

The distributed cache is a facility for caching large, application-specific files and distributing them efficiently across the nodes in a cluster.

> **See Also:** The description of `org.apache.hadoop.filecache.DistributedCache` in the Java documentation at
>
> `http://hadoop.apache.org/`

oracle.hadoop.loader.output.dirpathBufsize
Type: Integer

Default Value: `131072` (128 KB)

Description: Sets the size in bytes of the direct path stream buffer for `OCIOutputFormat`. Values are rounded up to the next multiple of 8 KB.

oracle.hadoop.loader.output.escapeEnclosers
Type: Boolean

Default Value: `false`

Description: Controls whether the embedded trailing encloser character is handled as literal text (that is, escaped). Set this property to `true` when a field may contain the trailing enclosure character as part of the data value. See `oracle.hadoop.loader.output.trailingFieldEncloser`.

oracle.hadoop.loader.output.fieldTerminator
Type: String

Default Value: , (comma)

Description: A character that indicates the end of an output field for `DelimitedTextInputFormat`. The value can be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding.

oracle.hadoop.loader.output.granuleSize
Type: Integer

Default Value: `10240000`

Description: The granule size in bytes for generated Data Pump files.

A granule determines the work load for a parallel process (PQ slave) when loading a file through the `ORACLE_DATAPUMP` access driver.

> **See Also:** *Oracle Database Utilities* for more information about the `ORACLE_DATAPUMP` access driver.

oracle.hadoop.loader.output.initialFieldEncloser
Type: String

Default Value: Not defined

Description: A character generated in the output to identify the beginning of a field. The value must be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding. A zero-length value means that no enclosers are generated in the output (default value).

Use this property when a field may contain the value of `oracle.hadoop.loader.output.fieldTerminator`. If a field may also contain the value of `oracle.hadoop.loader.output.trailingFieldEncloser`, then set `oracle.hadoop.loader.output.escapeEnclosers` to `true`.

If you set this property, then you must also set `oracle.hadoop.loader.output.trailingFieldEncloser`.

oracle.hadoop.loader.output.trailingFieldEncloser
Type: String

Default Value: Value of `oracle.hadoop.loader.output.initialFieldEncloser`

Description: A character generated in the output to identify the end of a field. The value must be either a single character or \u*HHHH*, where *HHHH* is the character's UTF-16 encoding. A zero-length value means that there are no enclosers (default value).

Use this property when a field may contain the value of `oracle.hadoop.loader.output.fieldTerminator`. If a field may also contain the value of `oracle.hadoop.loader.output.trailingFieldEncloser`, then set `oracle.hadoop.loader.output.escapeEnclosers` to `true`.

If you set this property, then you must also set `oracle.hadoop.loader.output.initialFieldEncloser`.

oracle.hadoop.loader.rejectLimit
Type: Integer

Default Value: `1000`

Description: The maximum number of rejected or skipped records allowed before the job stops running. A negative value turns off the reject limit and allows the job to run to completion.

If `mapred.map.tasks.speculative.execution` is `true` (the default), then the number of rejected records may be inflated temporarily, causing the job to stop prematurely.

Input format errors do not count toward the reject limit because they are fatal and cause the map task to stop. Errors encountered by the sampling feature or the online output formats do not count toward the reject limit either.

oracle.hadoop.loader.sampler.enableSampling
Type: Boolean

Default Value: `true`

Description: Controls whether the sampling feature is enabled. Set this property to `false` to disable sampling.

Even when `enableSampling` is set to `true`, the loader automatically disables sampling if it is unnecessary, or if the loader determines that a good sample cannot be made. For example, the loader disables sampling if the table is not partitioned, the number of reducer tasks is less than two, or there is too little input data to compute a good load balance. In these cases, the loader returns an informational message.

oracle.hadoop.loader.sampler.hintMaxSplitSize
Type: Integer

Default Value: `1048576` (1 MB)

Description: Sets the Hadoop `mapred.max.split.size` property for the sampling process; the value of `mapred.max.split.size` does not change for the job configuration. A value less than `1` is ignored.

Some input formats (such as `FileInputFormat`) use this property as a hint to determine the number of splits returned by `getSplits`. Smaller values imply that more chunks of data are sampled at random, which results in a better sample.

Increase this value for data sets with tens of terabytes of data, or if the input format `getSplits` method throws an out-of-memory error.

Although large splits are better for I/O performance, they are not necessarily better for sampling. Set this value small enough for good sampling performance, but no smaller. Extremely small values can cause inefficient I/O performance, and can cause `getSplits` to run out of memory by returning too many splits.

The `org.apache.hadoop.mapreduce.lib.input.FileInputFormat` method always returns splits at least as large as the minimum split size setting, regardless of the value of this property.

oracle.hadoop.loader.sampler.hintNumMapTasks

Type: Integer

Default Value: `100`

Description: Sets the value of the Hadoop `mapred.map.tasks` configuration property for the sampling process; the value of `mapred.map.tasks` does not change for the job configuration. A value less than `1` is ignored.

Some input formats (such as `DBInputFormat`) use this property as a hint to determine the number of splits returned by the `getSplits` method. Higher values imply that more chunks of data are sampled at random, which results in a better sample.

Increase this value for data sets with more than a million rows, but remember that extremely large values can cause `getSplits` to run out of memory by returning too many splits.

oracle.hadoop.loader.sampler.loadCI

Type: Decimal

Default Value: `0.95`

Description: The statistical confidence indicator for the maximum reducer load factor.

This property accepts values greater than or equal to `0.5` and less than `1` (`0.5 <=` *value* `< 1`). A value less than `0.5` resets the property to the default value. Typical values are `0.90`, `0.95`, and `0.99`.

See `oracle.hadoop.loader.sampler.maxLoadFactor`.

oracle.hadoop.loader.sampler.maxHeapBytes

Type: Integer

Default Value: `-1`

Description: Specifies in bytes the maximum amount of memory available to the sampler.

Sampling stops when one of these conditions is true:

- The sampler has collected the minimum number of samples required for load balancing.
- The percent of sampled data exceeds `oracle.hadoop.loader.sampler.maxSamplesPct`.
- The number of sampled bytes exceeds `oracle.hadoop.loader.sampler.maxHeapBytes`. This condition is not imposed when the property is set to a negative value.

oracle.hadoop.loader.sampler.maxLoadFactor

Type: Float

Default Value: `0.05` (5%)

Description: The maximum acceptable load factor for a reducer. A value of `0.05` indicates that reducers can be assigned up to 5% more data than their ideal load.

This property accepts values greater than `0`. A value less than or equal to 0 resets the property to the default value. Typical values are `0.05` and `0.1`.

In a perfectly balanced load, every reducer is assigned an equal amount of work (or load). The load factor is the relative overload for each reducer, calculated as (*assigned_load - ideal_load*)/*ideal_load*. If load balancing is successful, the job runs within the maximum load factor at the specified confidence.

See `oracle.hadoop.loader.sampler.loadCI`.

oracle.hadoop.loader.sampler.maxSamplesPct

Type: Float

Default Value: `0.01` (1%)

Description: Sets the maximum sample size as a fraction of the number of records in the input data. A value of `0.05` indicates that the sampler never samples more than 5% of the total number of records.

This property accepts a range of 0 to 1 (0% to 100%). A negative value disables it.

Sampling stops when one of these conditions is true:

- The sampler has collected the minimum number of samples required for load balancing, which can be fewer than the number set by this property.
- The percent of sampled data exceeds `oracle.hadoop.loader.sampler.maxSamplesPct`.
- The number of sampled bytes exceeds `oracle.hadoop.loader.sampler.maxHeapBytes`. This condition is not imposed when the property is set to a negative value.

oracle.hadoop.loader.sampler.minSplits

Type: Integer

Default Value: `5`

Description: The minimum number of input splits that the sampler reads from before it makes any evaluation of the stopping condition. If the total number of input splits is less than `minSplits`, then the sampler reads from all the input splits.

A number less than or equal to `0` is the same as a value of `1`.

oracle.hadoop.loader.sampler.numThreads

Type: Integer

Default Value: `5`

Description: The number of sampler threads. A higher number of threads allows higher concurrency in sampling. A value of `1` disables multithreading for the sampler.

Set the value based on the processor and memory resources available on the node where you start the Oracle Loader for Hadoop job.

oracle.hadoop.loader.sharedLibs

Type: String

Default Value:

```
${oracle.hadoop.loader.olh_home}/lib/libolh11.so,
${oracle.hadoop.loader.olh_home}/lib/libclntsh.so.11.1,
${oracle.hadoop.loader.olh_home}/lib/libnnz11.so,
${oracle.hadoop.loader.olh_home}/lib/libociei.so
```

Description: A comma-delimited list of library files, which is appended to the value of the `-files` command-line argument. You can copy your application files to the Hadoop cluster by using this property either instead of, or with, the `-files` option.

A leading comma or consecutive commas are invalid.

oracle.hadoop.loader.sortKey
Type: String

Default Value: Not defined

Description: A comma-delimited list of column names that forms a key for sorting output records within a reducer group.

The column names can be quoted or unquoted identifiers:

- A quoted identifier begins and ends with double quotation marks (`"`).
- An unquoted identifier is converted to uppercase before use.

oracle.hadoop.loader.tableMetadataFile
Type: String

Default Value: Not defined

Description: Path to the target table metadata file. Set this property when running in offline database mode.

Use the `file://` syntax to specify a local file, for example:

```
file:///home/jdoe/metadata.xml
```

To create the table metadata file, run the `OraLoaderMetadata` utility. See "OraLoaderMetadata Utility" on page 3-9.

oracle.hadoop.loader.targetTable
Type: String

Default Value: Not defined

Description: A schema-qualified name for the table to be loaded. Use this option to load all columns of the table when the names of the input fields match the column names. If you do not qualify the table name with the schema, then Oracle Loader for Hadoop uses the schema of the connection user.

For each column in the target table, the loader attempts to discover an input field with the same name as the column. The values from the matching field are loaded into the column.

This property takes priority over the `oracle.hadoop.loader.loaderMapFile` property.

Third-Party Licenses for Bundled Software

Oracle Loader for Hadoop installs the following third-party products:

- Apache Avro
- Apache Commons Mathematics Library
- Jackson JSON Processor

Oracle Loader for Hadoop includes Oracle 11*g* Release 2 (11.2) client libraries. For information about third party product included with Oracle Database 11*g* Release 2 (11.2), refer to*Oracle Database Licensing Information*.

Unless otherwise specifically noted, or as required under the terms of the third party license (e.g., LGPL), the licenses and statements herein, including all statements regarding Apache-licensed code, are intended as notices only.

Apache Licensed Code

The following is included as a notice in compliance with the terms of the Apache 2.0 License, and applies to all programs licensed under the Apache 2.0 license:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

A copy of the license is also reproduced below.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions**

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. **Grant of Patent License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution**. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   a. You must give any other recipients of the Work or Derivative Works a copy of this License; and

   b. You must cause any modified files to carry prominent notices stating that You changed the files; and

   c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the

attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions**. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. **Trademarks**. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. **Disclaimer of Warranty**. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability**. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability**. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Do not include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed by The Apache Software Foundation (http://www.apache.org/) (listed below):

Apache Avro 1.7.3

Licensed under the Apache License, Version 2.0 (the "License"); you may not use Apache Avro except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache Commons Mathematics Library 2.2

Copyright 2001-2011 The Apache Software Foundation

Licensed under the Apache License, Version 2.0 (the "License"); you may not use the Apache Commons Mathematics library except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Jackson JSON 1.8.8

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this library except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

4

Oracle Data Integrator Application Adapter for Hadoop

This chapter describes how to use the knowledge modules in Oracle Data Integrator (ODI) Application Adapter for Hadoop. It contains the following sections:

> **See Also:** *Oracle Fusion Middleware Application Adapters Guide for Oracle Data Integrator*

Introduction

Apache Hadoop is designed to handle and process data that is typically from data sources that are nonrelational and data volumes that are beyond what is handled by relational databases.

Oracle Data Integrator (ODI) Application Adapter for Hadoop enables data integration developers to integrate and transform data easily within Hadoop using Oracle Data Integrator. Employing familiar and easy-to-use tools and preconfigured knowledge modules (KMs), the application adapter provides the following capabilities:

- Loading data into Hadoop from the local file system and HDFS
- Performing validation and transformation of data within Hadoop
- Loading processed data from Hadoop to an Oracle database for further processing and generating reports

Knowledge modules (KMs) contain the information needed by Oracle Data Integrator to perform a specific set of tasks against a specific technology. An application adapter is a group of knowledge modules. Thus, Oracle Data Integrator Application Adapter for Hadoop is a group of knowledge modules for accessing data stored in Hadoop.

Concepts

Typical processing in Hadoop includes data validation and transformations that are programmed as MapReduce jobs. Designing and implementing a MapReduce job requires expert programming knowledge. However, when you use Oracle Data

Integrator and Oracle Data Integrator Application Adapter for Hadoop, you do not need to write MapReduce jobs. Oracle Data Integrator uses Hive and the Hive Query Language (HiveQL), a SQL-like language for implementing MapReduce jobs.

When you implement a big data processing scenario, the first step is to load the data into Hadoop. The data source is typically in the local file system, HDFS, Hive tables, or external Hive tables.

After the data is loaded, you can validate and transform it by using HiveQL like you use SQL. You can perform data validation (such as checking for NULLS and primary keys), and transformations (such as filtering, aggregations, set operations, and derived tables). You can also include customized procedural snippets (scripts) for processing the data.

When the data has been aggregated, condensed, or processed into a smaller data set, you can load it into an Oracle database for further processing and analysis. Oracle Loader for Hadoop is recommended for optimal loading into an Oracle database.

Knowledge Modules

Oracle Data Integrator provides the knowledge modules (KMs) described in Table 4–1 for use with Hadoop.

Table 4–1 Oracle Data Integrator Application Adapter for Hadoop Knowledge Modules

KM Name	Description	Source	Target
IKM File to Hive (Load Data)	Loads data from local and HDFS files into Hive tables. It provides options for better performance through Hive partitioning and fewer data movements. This knowledge module supports wildcards (*,?).	File system	Hive
IKM Hive Control Append	Integrates data into a Hive target table in truncate/insert (append) mode. Data can be controlled (validated). Invalid data is isolated in an error table and can be recycled.	Hive	Hive
IKM Hive Transform	Integrates data into a Hive target table after the data has been transformed by a customized script such as Perl or Python	Hive	Hive
IKM File-Hive to Oracle (OLH)	Integrates data from an HDFS file or Hive source into an Oracle database target using Oracle Loader for Hadoop, Oracle SQL Connector for HDFS, or both.	File system or Hive	Oracle Database
CKM Hive	Validates data against constraints	NA	Hive
RKM Hive	Reverse engineers Hive tables	Hive metadata	NA

Security

For security information for Oracle Data Integrator, see the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Setting Up the Topology

To set up the topology in Oracle Data Integrator, you identify the data server and the physical and logical schemas that are used to store the file system and Hive information.

This section contains the following topics:

- Setting Up File Data Sources
- Setting Up Hive Data Sources
- Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs
- Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent

> **Note:** Many of the environment variables described in the following sections are already configured for Oracle Big Data Appliance. See the configuration script at /opt/oracle/odiagent-*version*/agent_standalone/oracledi/agent/bin/HadoopEnvSetup.sh

Setting Up File Data Sources

In the Hadoop context, there is a distinction between files in Hadoop Distributed File System (HDFS) and local files (files outside of HDFS).

To define a data source:

1. Create a DataServer object under File technology.
2. Create a Physical Schema object for every directory to be accessed.
3. Create a Logical Schema object for every directory to be accessed.
4. Create a Model for every Logical Schema.
5. Create one or more data stores for each different type of file and wildcard name pattern.
6. For HDFS files, create a DataServer object under File technology by entering the HDFS name node in the field JDBC URL. For example:

```
hdfs://bda1node01.example.com:8020
```

> **Note:** No dedicated technology is defined for HDFS files.

Setting Up Hive Data Sources

The following steps in Oracle Data Integrator are required for connecting to a Hive system. Oracle Data Integrator connects to Hive by using JDBC.

Prerequisites

The Hive technology must be included in the standard Oracle Data Integrator technologies. If it is not, then import the technology in `INSERT_UPDATE` mode from the xml-reference directory.

You must add all Hive-specific flex fields. For pre-11.1.1.6.0 repositories, the flex fields are added during the repository upgrade process.

To set up a Hive data source:

1. Ensure that the following environment variables are set, and note their values. The following list shows typical values, although your installation may be different:
   - $HIVE_HOME: /usr/lib/hive
   - $HADOOP_HOME: /usr/lib/hadoop (contains configuration files such as core-site.xml)
   - $OSCH_HOME: /opt/oracle/orahdfs
2. Open ~/.odi/oracledi/userlib/additional_path.txt in a text editor and add the paths listed in Table 4–2. Enter the full path obtained in Step 1 instead of the variable name.

   This step enables ODI Studio to access the JAR files.

Table 4–2 JAR File Paths

Description	CDH4 Path	CDH3 Path
Hive JAR Files	*$HIVE_HOME*/lib/*.jar	*$HIVE_HOME*/*.jar
Hadoop Client JAR Files	*$HADOOP_HOME*/client/*.jar	*$HADOOP_HOME*/hadoop-*-core*.jar[1] *$HADOOP_HOME*/hadoop-*-tools*.jar[1]
Hadoop Configuration Directory	*$HADOOP_HOME*	*$HADOOP_HOME*
Oracle SQL Connector for HDFS JAR Files (optional)	*$OSCH_HOME*/jlib/*.jar	*$OSCH_HOME*/jlib/*.jar

[1] Replace the stars (*) with the full file name.

3. Ensure that the Hadoop configuration directory is in the ODI class path.

   The Hadoop configuration directory contains files such as core-default.xml, core-site.xml, and hdfs-site.xml.
4. Create a DataServer object under Hive technology.
5. Set the following locations under JDBC:

   JDBC Driver: `org.apache.hadoop.hive.jdbc.HiveDriver`

   JDBC URL: for example, `jdbc:hive://BDA:10000/default`
6. Set the following under Flexfields:

   Hive Metastore URIs: for example, `thrift://BDA:10000`
7. Create a Physical Default Schema.

   As of Hive 0.7.0, no schemas or databases are supported. Only Default is supported. Enter `default` in both schema fields of the physical schema definition.
8. Ensure that the Hive server is up and running.
9. Test the connection to the DataServer.
10. Create a Logical Schema object.
11. Create at least one Model for the LogicalSchema.
12. Import RKM Hive as a global knowledge module or into a project.

13. Create a new model for Hive Technology pointing to the logical schema.
14. Perform a custom reverse-engineering operation using RKM Hive.

At the end of this process, the Hive DataModel contains all Hive tables with their columns, partitioning, and clustering details stored as flex field values.

Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs

After setting up an Oracle Data Integrator agent, configure it to work with Oracle Data Integrator Application Adapter for Hadoop.

> **Note:** Many file names contain the version number. When you see a star (*) in a file name, check your installation and enter the full file name.

To configure the Oracle Data Integrator agent:

1. Install Hadoop on your Oracle Data Integrator agent computer. Ensure that the `HADOOP_HOME` environment variable is set.

   For Oracle Big Data Appliance, see the *Oracle Big Data Appliance Software User's Guide* for instructions for setting up a remote Hadoop client.
2. Install Hive on your Oracle Data Integrator agent computer. Ensure that the `HIVE_HOME` environment variable is set.
3. Ensure that the Hadoop configuration directory is in the ODI class path.

   The Hadoop configuration directory contains files such as core-default.xml, core-site.xml, and hdfs-site.xml.
4. Add paths to `ODI_ADDITIONAL_CLASSPATH`, so that the ODI agent can access the JAR files. If you are not using Oracle SQL Connector for HDFS, then omit $OSCH_HOME from the setting.

   > **Note:** In these commands, $HADOOP_CONF points to the directory containing the Hadoop configuration files. This directory is often the same as $HADOOP_HOME.

   - For CDH4, use a command like the following:

     ```
     ODI_ADDITIONAL_CLASSPATH=$HIVE_HOME/lib/'*':$HADOOP_HOME/client/'*':$OSCH_
     HOME/jlib/'*':$HADOOP_CONF
     ```

   - For CDH3, use a command like the following, replacing hadoop-*-core*.jar and hadoop-*-tools*.jar with the full path names:

     ```
     ODI_ADDITIONAL_CLASSPATH=$HIVE_HOME/lib/'*':$HADOOP_
     HOME/hadoop-*-core*.jar:$HADOOP_HOME/hadoop-*-tools*.jar:$OSCH_
     HOME/jlib/'*':$HADOOP_CONF
     ```

5. Set environment variable `ODI_HIVE_SESSION_JARS` to include Hive Regex SerDe:

   ```
   ODI_HIVE_SESSION_JARS=$HIVE_HOME/lib/hive-contrib-*.jar
   ```

   Include other JAR files as required, such as custom SerDes JAR files. These JAR files are added to every Hive JDBC session and thus are added to every Hive MapReduce job.

6. Set environment variable HADOOP_CLASSPATH:

```
HADOOP_CLASSPATH=$HIVE_HOME/lib/hive-metastore-*.jar:$HIVE_
HOME/lib/libthrift.jar:$HIVE_HOME/lib/libfb*.jar:$HIVE_
HOME/lib/hive-common-*.jar:$HIVE_HOME/lib/hive-exec-*.jar.
```

   This setting enables the Hadoop script to start Hive MapReduce jobs.

To use Oracle Loader for Hadoop:

1. Install Oracle Loader for Hadoop on your Oracle Data Integrator agent system. See "Installing Oracle Loader for Hadoop" on page 1-12.
2. Install Oracle Database client on your Oracle Data Integrator agent system. See "Oracle Loader for Hadoop Setup" on page 1-12 for the Oracle Database client version.
3. Set environment variable OLH_HOME.
4. Set environment variable ODI_OLH_JARS.

   You must list all JAR files required for Oracle Loader for Hadoop. See the oracle.hadoop.loader.libjars property on page 3-31.

   This is a comma-separated list of JAR files, which is formatted for readability. The list includes the Hive JAR files needed by Oracle Data Integrator Application Adapter for Hadoop. Enter valid file names for your installation.

   For example:

```
$HIVE_HOME/lib/hive-metastore-*.jar,
$HIVE_HOME/lib/libthrift.jar,
$HIVE_HOME/lib/libfb*.jar,
      .
      .
      .
```

5. Add paths to HADOOP_CLASSPATH:

```
HADOOP_CLASSPATH=$OLH_HOME/jlib/'*':$HADOOP_CLASSPATH
```

6. Set environment variable ODI_HIVE_SESSION_JARS to include Hive Regex SerDe:

```
ODI_HIVE_SESSION_JARS=$HIVE_HOME/lib/hive-contrib-*.jar
```

   Include other JAR files as required, such as custom SerDes JAR files. These JAR files are added to every Hive JDBC session and thus are added to every Hive MapReduce job.

7. Add any custom native libraries required for Oracle Loader for Hadoop. See the oracle.hadoop.loader.sharedLibs property on page 3-36. This is a comma-separated list.
8. To use Oracle SQL Connector for HDFS (OLH_OUTPUT_MODE=DP_OSCH or OSCH), you must first install it. See "Oracle SQL Connector for Hadoop Distributed File System Setup" on page 1-4.

Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent

For executing Hadoop jobs on the local agent of an Oracle Data Integrator Studio installation, follow the configuration steps in the previous section with the following

change: Copy JAR files into the Oracle Data Integrator userlib directory instead of the drivers directory.

Setting Up an Integration Project

Setting up a project follows the standard procedures. See the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Import the following KMs into Oracle Data Integrator project:

- IKM File to Hive (Load Data)
- IKM Hive Control Append
- IKM Hive Transform
- IKM File-Hive to Oracle (OLH)
- CKM Hive
- RKM Hive

Creating an Oracle Data Integrator Model from a Reverse-Engineered Hive Model

This section contains the following topics:

Creating a Model

To create a model that is based on the technology hosting Hive and on the logical schema created when you configured the Hive connection, follow the standard procedure described in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Reverse Engineering Hive Tables

RKM Hive is used to reverse engineer Hive tables and views. To perform a customized reverse-engineering of Hive tables with RKM Hive, follow the usual procedures, as described in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*. This topic details information specific to Hive tables.

The reverse-engineering process creates the data stores for the corresponding Hive table or views. You can use the data stores as either a source or a target in an integration interface.

RKM Hive

RKM Hive reverses these metadata elements:

- Hive tables and views as Oracle Data Integrator data stores.

  Specify the reverse mask in the Mask field, and then select the tables and views to reverse. The Mask field in the Reverse tab filters reverse-engineered objects based on their names. The Mask field cannot be empty and must contain at least the percent sign (%).

- Hive columns as Oracle Data Integrator columns with their data types.

- Information about buckets, partitioning, clusters, and sort columns are set in the respective flex fields in the Oracle Data Integrator data store or column metadata.

Table 4–3 describes the options for RKM Hive.

Table 4–3 RKM Hive Options

Option	Description
USE_LOG	Log intermediate results?
LOG_FILE_NAME	Path and file name of log file. Default path is the user home and the default file name is reverse.log.

Table 4–4 describes the created flex fields.

Table 4–4 Flex Fields for Reverse-Engineered Hive Tables and Views

Object	Flex Field Name	Flex Field Code	Flex Field Type	Description
DataStore	Hive Buckets	HIVE_BUCKETS	String	Number of buckets to be used for clustering
Column	Hive Partition Column	HIVE_PARTITION_COLUMN	Numeric	All partitioning columns are marked as "1". Partition information can come from the following: ▪ Mapped source column ▪ Constant value specified in the target column ▪ File name fragment
Column	Hive Cluster Column	HIVE_CLUSTER_COLUMN	Numeric	All cluster columns are marked as "1".
Column	Hive Sort Column	HIVE_SORT_COLUMN	Numeric	All sort columns are marked as "1".

Designing the Interface

After reverse engineering Hive tables and configuring them, you can choose from these interface configurations:

- Loading Data from Files into Hive
- Validating and Transforming Data Within Hive
- Loading Data into an Oracle Database from Hive and HDFS

Loading Data from Files into Hive

To load data from the local file system or the HDFS file system into Hive tables:

1. Create the data stores for local files and HDFS files.

   Refer to the *Oracle Fusion Middleware Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for information about reverse engineering and configuring local file data sources.

2. Create an interface using the file data store as the source and the corresponding Hive table as the target. Use the IKM File to Hive (Load Data) knowledge module specified in the flow tab of the interface. This integration knowledge module loads data from flat files into Hive, replacing or appending any existing data.

IKM File to Hive

IKM File to Hive (Load Data) supports:

- One or more input files. To load multiple source files, enter an asterisk or a question mark as a wildcard character in the resource name of the file DataStore (for example, webshop_*.log).
- File formats:
  - Fixed length
  - Delimited
  - Customized format
- Loading options:
  - Immediate or deferred loading
  - Overwrite or append
  - Hive external tables

Table 4–5 describes the options for IKM File to Hive (Load Data). See the knowledge module for additional details.

Table 4–5 IKM File to Hive Options

Option	Description
CREATE_TARG_TABLE	Create target table.
TRUNCATE	Truncate data in target table.
FILE_IS_LOCAL	Is the file in the local file system or in HDFS?
EXTERNAL_TABLE	Use an externally managed Hive table.
USE_STAGING_TABLE	Use a Hive staging table. Select this option if the source and target do not match or if the partition column value is part of the data file. If the partitioning value is provided by a file name fragment or a constant in target mapping, then set this value to `false`.
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after the interface execution.
DEFER_TARGET_LOAD	Load data into the final target now or defer?
OVERRIDE_ROW_FORMAT	Provide a parsing expression for handling a custom file format to perform the mapping from source to target.
STOP_ON_FILE_NOT_FOUND	Stop if no source file is found?

Validating and Transforming Data Within Hive

After loading data into Hive, you can validate and transform the data using the following knowledge modules.

IKM Hive Control Append

This knowledge module validates and controls the data, and integrates it into a Hive target table in truncate/insert (append) mode. Invalid data is isolated in an error table and can be recycled. IKM Hive Control Append supports inline view interfaces that use either this knowledge module or IKM Hive Transform.

Table 4–6 lists the options. See the knowledge module for additional details.

Table 4–6 IKM Hive Control Append Options

Option	Description
FLOW_CONTROL	Validate incoming data?
RECYCLE_ERRORS	Reintegrate data from error table?
STATIC_CONTROL	Validate data after load?
CREATE_TARG_TABLE	Create target table?
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
TRUNCATE	Truncate data in target table?

CKM Hive

This knowledge module checks data integrity for Hive tables. It verifies the validity of the constraints of a Hive data store and diverts the invalid records to an error table. You can use CKM Hive for static control and flow control. You must also define these constraints on the stored data.

Table 4–7 lists the options for this check knowledge module. See the knowledge module for additional details.

Table 4–7 CKM Hive Options

Option	Description
DROP_ERROR_TABLE	Drop error table before execution?

IKM Hive Transform

This knowledge module performs transformations. It uses a shell script to transform the data, and then integrates it into a Hive target table using replace mode. The knowledge module supports inline view interfaces and can be used as an inline-view for IKM Hive Control Append.

The transformation script must read the input columns in the order defined by the source data store. Only mapped source columns are streamed into the transformations. The transformation script must provide the output columns in the order defined by the target data store.

Table 4–8 lists the options for this integration knowledge module. See the knowledge module for additional details.

Table 4–8 IKM Hive Transform Options

Option	Description
CREATE_TARG_TABLE	Create target table?
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
TRANSFORM_SCRIPT_NAME	Script file name
TRANSFORM_SCRIPT	Script content
PRE_TRANSFORM_DISTRIBUTE	Provides an optional, comma-separated list of source column names, which enables the knowledge module to distribute the data before the transformation script is applied
PRE_TRANSFORM_SORT	Provide an optional, comma-separated list of source column names, which enables the knowledge module to sort the data before the transformation script is applied

Table 4–8 (Cont.) IKM Hive Transform Options

Option	Description
POST_TRANSFORM_DISTRIBUTE	Provides an optional, comma-separated list of target column names, which enables the knowledge module to distribute the data after the transformation script is applied
POST_TRANSFORM_SORT	Provides an optional, comma-separated list of target column names, which enables the knowledge module to sort the data after the transformation script is applied

Loading Data into an Oracle Database from Hive and HDFS

IKM File-Hive to Oracle (OLH) integrates data from an HDFS file or Hive source into an Oracle database target using Oracle Loader for Hadoop. Using the interface configuration and the selected options, the knowledge module generates an appropriate Oracle Database target instance. Hive and Hadoop versions must follow the Oracle Loader for Hadoop requirements.

See Also:

- "Oracle Loader for Hadoop Setup" on page 1-12 for required versions of Hadoop and Hive
- "Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs" on page 4-5 for required environment variable settings

Table 4–9 lists the options for this integration knowledge module. See the knowledge module for additional details.

Table 4–9 IKM File - Hive to Oracle (OLH)

Option	Description
OLH_OUTPUT_MODE	Specify JDBC, OCI, or Data Pump for data transfer.
CREATE_TARG_TABLE	Create target table?
REJECT_LIMIT	Maximum number of errors for Oracle Loader for Hadoop and EXTTAB.
USE_HIVE_STAGING_TABLE	Materialize Hive source data before extract?
USE_ORACLE_STAGING_TABLE	Use an Oracle database staging table?
EXT_TAB_DIR_LOCATION	Shared file path used for Oracle Data Pump transfer.
TEMP_DIR	Local path for temporary files.
MAPRED_OUTPUT_BASE_DIR	HDFS directory for Oracle Loader for Hadoop output files.
FLOW_TABLE_OPTIONS	Options for flow (stage) table creation when you are using an Oracle database staging table.
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
OVERRIDE_INPUTFORMAT	Set to handle custom file formats.
EXTRA_OLH_CONF_PROPERTIES	Optional Oracle Loader for Hadoop configuration file properties
TRUNCATE	Truncate data in target table?
DELETE_ALL	Delete all data in target table?

5

Oracle R Connector for Hadoop

This chapter describes R support for big data. It contains the following sections:

About Oracle R Connector for Hadoop

Oracle R Connector for Hadoop is a collection of R packages that provide:

- Interfaces to work with Hive tables, the Apache Hadoop compute infrastructure, the local R environment, and Oracle database tables
- Predictive analytic techniques, written in R or Java as Hadoop MapReduce jobs, that can be applied to data in HDFS files

You install and load this package as you would any other R package. Using simple R functions, you can perform tasks like these:

- Access and transform HDFS data using a Hive-enabled transparency layer
- Use the R language for writing mappers and reducers
- Copy data between R memory, the local file system, HDFS, Hive, and Oracle databases
- Schedule R programs to execute as Hadoop MapReduce jobs and return the results to any of those locations

Several analytic algorithms are available in Oracle R Connector for Hadoop: linear regression, neural networks for prediction, matrix completion using low rank matrix factorization, clustering, and non-negative matrix factorization. They are written in either Java or R.

To use Oracle R Connector for Hadoop, you should be familiar with MapReduce programming, R programming, and statistical methods.

Oracle R Connector for Hadoop APIs

Oracle R Connector for Hadoop provides access from a local R client to Apache Hadoop using functions with these prefixes:

- `hadoop`: Identifies functions that provide an interface to Hadoop MapReduce
- `hdfs`: Identifies functions that provide an interface to HDFS
- `orch`: Identifies a variety of functions; `orch` is a general prefix for `ORCH` functions
- `ore`: Identifies functions that provide an interface to a Hive data store

Oracle R Connector for Hadoop uses data frames as the primary object type, but it can also operate on vectors and matrices to exchange data with HDFS. The APIs support the numeric, integer, and character data types in R.

All of the APIs are included in the `ORCH` library. The functions are listed in Chapter 6 in alphabetical order.

See Also: The R Project website at `http://www.r-project.org/`

Access to HDFS Files

For Oracle R Connector for Hadoop to access the data stored in HDFS, the input files must comply with the following requirements:

- All input files for a MapReduce job must be stored in one directory as the parts of one logical file. Any valid HDFS directory name and file name extensions are acceptable.
- Any file in that directory with a name beginning with an underscore (_) is ignored.
- The input files must be in comma-separated value (CSV) format as follows:
  - *key* := *string* | ""
  - *value* := *string*[,*value*]
  - *line* := [*key*\t]*value*
  - *string* := *char*[*string*]
  - *char* := *regexp*([^,\n])

Input File Format Examples

The following are examples of acceptable CSV input files:

- CSV files with a key:

```
"1\tHello,world"
"2\tHi,there"
```

- CSV files with no key:

```
"Hello,world"
"Hi,there"
```

- CSV files with a NULL key:

```
"\tHello,world"
"\tHi,there"
```

Access to Hive

Hive provides an alternative storage and retrieval mechanism to HDFS files through a querying language called HiveQL, which closely resembles SQL. Hive uses MapReduce for distributed processing. However, the data is structured and has additional metadata to support data discovery. Oracle R Connector for Hadoop uses the data preparation and analysis features of HiveQL, while enabling you to use R language constructs.

See Also: The Apache Hive website at `http://hive.apache.org`

ORE Functions for Hive

You can connect to Hive and manage objects using R functions that have an `ore` prefix, such as `ore.connect`. If you are also using Oracle R Enterprise, then you recognize these functions. The `ore` functions in Oracle R Enterprise create and manage objects in an Oracle database, and the `ore` functions in Oracle R Connector for Hadoop create and manage objects in a Hive database. You can connect to one database at a time, either Hive or Oracle Database, but not both simultaneously.

The following ORE functions are supported in Oracle R Connector for Hadoop:

```
as.ore
as.ore.character
as.ore.frame
as.ore.integer
as.ore.logical
as.ore.numeric
as.ore.vector
is.ore
is.ore.character
is.ore.frame
is.ore.integer
is.ore.logical
is.ore.numeric
is.ore.vector
ore.create
ore.drop
ore.get
ore.pull
ore.push
ore.recode
```

This release does not support `ore.factor`, `ore.list`, or `ore.object`.

These aggregate functions from OREStats are also supported:

```
aggregate
fivenum
IQR
median
quantile
sd
var*
```

*For vectors only

Generic R Functions Supported in Hive

Oracle R Connector for Hadoop also overloads the following standard generic R functions with methods to work with Hive objects.

Character methods

`casefold`, `chartr`, `gsub`, `nchar`, `substr`, `substring`, `tolower`, `toupper`

This release does not support `grepl` or `sub`.

Frame methods

- `attach`, `show`
- `[`, `$`, `$<-`, `[[`, `[[<-`
- Subset functions: `head`, `tail`
- Metadata functions: `dim`, `length`, `NROW`, `nrow`, `NCOL`, `ncol`, `names`, `names<-`, `colnames`, `colnames<-`
- Conversion functions: `as.data.frame`, `as.env`, `as.list`
- Arithmetic operators: `+`, `-`, `*`, `^`, `%%`, `%/%`, `/`
- `Compare`, `Logic`, `xor`, `!`
- Test functions: `is.finite`, `is.infinite`, `is.na`, `is.nan`
- Mathematical transformations: `abs`, `acos`, `asin`, `atan`, `ceiling`, `cos`, `exp`, `expm1`, `floor`, `log`, `log10`, `log1p`, `log2`, `logb`, `round`, `sign`, `sin`, `sqrt`, `tan`, `trunc`
- Basic statistics: `colMeans`, `colSums`, `rowMeans`, `rowSums`, `Summary`, `summary`, `unique`
- `by`, `merge`
- `unlist`, `rbind`, `cbind`, `data.frame`, `eval`

This release does not support `dimnames`, `interaction`, `max.col`, `row.names`, `row.names<-`, `scale`, `split`, `subset`, `transform`, `with`, or `within`.

Logical methods

`ifelse`, `Logic`, `xor`, `!`

Matrix methods

Not supported

Numeric methods

- Arithmetic operators: `+`, `-`, `*`, `^`, `%%`, `%/%`, `/`
- Test functions: `is.finite`, `is.infinite`, `is.nan`
- `abs`, `acos`, `asin`, `atan`, `ceiling`, `cos`, `exp`, `expm1`, `floor`, `log`, `log1p`, `log2`, `log10`, `logb`, `mean`, `round`, `sign`, `sin`, `sqrt`, `Summary`, `summary`, `tan`, `trunc`, `zapsmall`

This release does not support `atan2`, `besselI`, `besselK`, `besselJ`, `besselY`, `diff`, `factorial`, `lfactorial`, `pmax`, `pmin`, or `tabulate`.

Vector methods

- `show`, `length`, `c`
- Test functions: `is.vector`, `is.na`
- Conversion functions: `as.vector`, `as.character`, `as.numeric`, `as.integer`, `as.logical`
- `[`, `[<-`, `|`

- by, Compare, head, %in%, paste, sort, table, tail, tapply, unique

This release does not support interaction, lengthb, rank, or split.

Example 5–1 shows simple data preparation and processing. For additional details, see "Support for Hive Data Types" on page 5-5.

Example 5–1 Using R to Process Data in Hive Tables

```
# Connect to Hive
ore.connect(type="HIVE")

# Attach the current envt. into search path of R
ore.attach()

# create a Hive table by pushing the numeric columns of the iris data set
IRIS_TABLE <- ore.push(iris[1:4])

# Create bins based on Petal Length
 IRIS_TABLE$PetalBins = ifelse(IRIS_TABLE$Petal.Length < 2.0, "SMALL PETALS",
+                        ifelse(IRIS_TABLE$Petal.Length < 4.0, "MEDIUM PETALS",
+                        ifelse(IRIS_TABLE$Petal.Length < 6.0,
+                               "MEDIUM LARGE PETALS", "LARGE PETALS")))

#PetalBins is now a derived column of the HIVE object
> names(IRIS_TABLE)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "PetalBins"

# Based on the bins, generate summary statistics for each group
aggregate(IRIS_TABLE$Petal.Length, by = list(PetalBins = IRIS_TABLE$PetalBins),
+           FUN = summary)
1        LARGE PETALS    6 6.025000 6.200000 6.354545 6.612500  6.9    0
2 MEDIUM LARGE PETALS    4 4.418750 4.820000 4.888462 5.275000  5.9    0
3       MEDIUM PETALS    3 3.262500 3.550000 3.581818 3.808333  3.9    0
4        SMALL PETALS    1 1.311538 1.407692 1.462000 1.507143  1.9    0
Warning message:
ORE object has no unique key - using random order
```

Support for Hive Data Types

Oracle R Connector for Hadoop can access any Hive table containing columns with string and numeric data types such as tinyint, smallint, bigint, int, float, and double. There is no support for these complex data types:

```
array
binary
map
struct
timestamp
union
```

If you attempt to access a Hive table containing an unsupported data type, then you get an error message. To access the table, you must convert the column to a supported data type.

To convert a column to a supported data type:

1. Open the Hive command interface:

```
$ hive
hive>
```

2. Identify the column with an unsupported data type:

```
hive> describe table_name;
```

3. View the data in the column:

```
hive> select column_name from table_name;
```

4. Create a table for the converted data, using only supported data types.
5. Copy the data into the new table, using an appropriate conversion tool.

Example 5–2 shows the conversion of an array. Example 5–3 and Example 5–4 show the conversion of timestamp data.

Example 5–2 Converting an Array to String Columns

```
R> ore.sync(table="t1")
   Warning message:
   table t1 contains unsupported data types
     .
     .
     .
hive> describe t1;
OK
      col1   int
      col2   array<string>

hive> select * from t1;
OK
1      ["a","b","c"]
2      ["d","e","f"]
3      ["g","h","i"]

hive> create table t2 (c1 string, c2 string, c2 string);
hive> insert into table t2 select col2[0], col2[1], col2[2] from t1;
     .
     .
     .
R> ore.sync(table="t2")
R> ore.ls()
[1] "t2"
R> t2$c1
[1] "a" "d" "g"
```

Example 5–3 uses automatic conversion of the `timestamp` data type into string. The data is stored in a table named `t5` with a column named `tstmp`.

Example 5–3 Converting a Timestamp Column

```
hive> select * from t5;


hive> create table t6 (timestmp string);
hive> insert into table t6 SELECT tstmp from t5;
```

Example 5–4 uses the Hive `get_json_object` function to extract the two columns of interest from the JSON table into a separate table for use by Oracle R Connector for Hadoop.

Example 5–4 Converting a Timestamp Column in a JSON File

```
hive> select * from t3;
OK

{"custId":1305981,"movieId":null,"genreId":null,"time":"2010-12-30:23:59:32","reco
mmended":null,"activity":9}

hive> create table t4 (custid int, time string);

hive> insert into table t4 SELECT cast(get_json_object(c1, '$.custId') as int),
cast(get_json_object(c1, '$.time') as string) from t3;
```

Usage Notes for Hive Access

The Hive command language interface (CLI) is used for executing queries and provides support for Linux clients. There is no JDBC or ODBC support.

The `ore.create` function creates Hive tables only as text files. However, Oracle R Connector for Hadoop can access Hive tables stored as either text files or sequence files.

You can use the `ore.exec` function to execute Hive commands from the R console. For a demo, run the `hive_sequencefile` demo.

Oracle R Connector for Hadoop can access tables and views in the default Hive database only. To allow read access to objects in other databases, you must expose them in the default database. For example, you can create views.

Oracle R Connector for Hadoop does not have a concept of ordering in Hive. An R frame persisted in Hive might not have the same ordering after it is pulled out of Hive and into memory. Oracle R Connector for Hadoop is designed primarily to support data cleanup and filtering of huge HDFS data sets, where ordering is not critical. You might see warning messages when working with unordered Hive frames:

```
Warning messages:
1: ORE object has no unique key - using random order
2: ORE object has no unique key - using random order
```

To suppress these warnings, set the `ore.warn.order` option in your R session:

```
R> options(ore.warn.order = FALSE)
```

Example: Loading Hive Tables into Oracle R Connector for Hadoop

Table 6–1 provides an example of loading a Hive table into an R data frame for analysis. It uses these Oracle R Connector for Hadoop functions:

```
hdfs.attach
ore.attach
ore.connect
ore.create
ore.hiveOptions
ore.sync
```

Example 5–5 Loading a Hive Table

```
# Connect to HIVE metastore and sync the HIVE input table into the R session.
ore.connect(type="HIVE")
ore.sync(table="datatab")
ore.attach()
```

```
# The "datatab" object is a Hive table with columns named custid, movieid,
activity, and rating.
# Perform filtering to remove missing (NA) values from custid and movieid columns
# Project out three columns: custid, movieid and rating
t1 <- datatab[!is.na(datatab$custid) &
    !is.na(datatab$movieid) &
    datatab$activity==1, c("custid","movieid", "rating")]

# Set HIVE field delimiters to ','. By default, it is Ctrl+a for text files but
# ORCH 2.0 supports only ',' as a file separator.
ore.hiveOptions(delim=',')

# Create another Hive table called "datatab1" after the transformations above.
ore.create (t1, table="datatab1")

# Use the HDFS directory, where the table data for datatab1 is stored, to attach
# it to ORCH framework. By default, this location is "/user/hive/warehouse"
dfs.id <- hdfs.attach("/user/hive/warehouse/datatab1")

# dfs.id can now be used with all hdfs.*, orch.* and hadoop.* APIs of ORCH for
further processing and analytics.
```

Access to Oracle Database

Oracle R Connector for Hadoop provides a basic level of database access. You can move the contents of a database table to HDFS, and move the results of HDFS analytics back to the database.

You can then perform additional analysis on this smaller set of data using a separate product named Oracle R Enterprise. It enables you to perform statistical analysis on database tables, views, and other data objects using the R language. You have transparent access to database objects, including support for Business Intelligence and in-database analytics.

Access to the data stored in an Oracle database is always restricted to the access rights granted by your DBA.

Oracle R Enterprise is included in the Oracle Advanced Analytics option to Oracle Database Enterprise Edition. It is not one of the Oracle Big Data Connectors.

See Also: *Oracle R Enterprise User's Guide*

Usage Notes for Oracle Database Access

Oracle R Connector for Hadoop uses Sqoop to move data between HDFS and Oracle Database. Sqoop imposes several limitations on Oracle R Connector for Hadoop:

- You cannot import Oracle tables with `BINARY_FLOAT` or `BINARY_DOUBLE` columns. As a work-around, you can create a view that casts these columns to `NUMBER`.
- All column names must be in upper case.

Scenario for Using Oracle R Connector for Hadoop with Oracle R Enterprise

The following scenario may help you identify opportunities for using Oracle R Connector for Hadoop with Oracle R Enterprise.

Using Oracle R Connector for Hadoop, you can look for files that you have access to on HDFS and execute R calculations on data in one such file. You can also upload data stored in text files on your local file system into HDFS for calculations, schedule an R

script for execution on the Hadoop cluster using DBMS_SCHEDULER, and download the results into a local file.

Using Oracle R Enterprise, you can open the R interface and connect to Oracle Database to work on the tables and views that are visible based on your database privileges. You can filter out rows, add derived columns, project new columns, and perform visual and statistical analysis.

Again using Oracle R Connector for Hadoop, you might deploy a MapReduce job on Hadoop for CPU-intensive calculations written in R. The calculation can use data stored in HDFS or, with Oracle R Enterprise, in an Oracle database. You can return the output of the calculation to an Oracle database and to the R console for visualization or additional processing.

Analytic Functions in Oracle R Connector for Hadoop

Table 5–1 describes the analytic functions. For more information, use R Help.

Table 5–1 Descriptions of the Analytic Functions

Function	Description
orch.evaluate	Evaluates a fit generated by orch.lmf. This information can be helpful when you are tuning the model parameters.
orch.export.fit	Exports a model (W and H factor matrices) to the specified destination for orch.lmf.jellyfish or orch.nmf. It is not used for orch.mahout.lmf.als.
orch.lm	Fits a linear model using tall-and-skinny QR (TSQR) factorization and parallel distribution. The function computes the same statistical parameters as the Oracle R Enterprise ore.lm function.
orch.lmf	Fits a low rank matrix factorization model using either the jellyfish algorithm or the Mahout alternating least squares with weighted regularization (ALS-WR) algorithm.
orch.neural	Provides a neural network to model complex, nonlinear relationships between inputs and outputs, or to find patterns in the data.
orch.nmf	Provides the main entry point to create a nonnegative matrix factorization model using the jellyfish algorithm. This function can work on much larger data sets than the R NMF package, because the input does not need to fit into memory.
orch.nmf.NMFalgo	Plugs in to the R NMF package framework as a custom algorithm. This function is used for benchmark testing.
orch.recommend	Computes the top *n* items to be recommended for each user that has predicted ratings based on the input orch.mahout.lmf.asl model.
predict.orch.lm	Predicts future results based on the fit calculated by orch.lm.
print.orch.lm	Prints a model returned by the orch.lmf function.
print.summary.orch.lm	Prints a summary of the fit calculated by orch.lm.
summary.orch.lm	Prepares a summary of the fit calculated by orch.lm.

ORCH mapred.config Class

The hadoop.exec and hadoop.run functions have an optional argument, config, for configuring the resultant MapReduce job. This argument is an instance of the mapred.config class.

The `mapred.config` class has these slots:

hdfs.access
Set to `TRUE` to allow access to the `HDFS.*` functions in the mappers, reducers, and combiners, or set to `FALSE` to restrict access (default).

job.name
A descriptive name for the job so that you can monitor its progress more easily.

map.input
A mapper input data-type keyword: `data.frame`, `list`, or `vector` (default).

map.output
A sample R data frame object that defines the output structure of data from the mappers. It is required only if the mappers change the input data format. Then the reducers require a sample data frame to parse the input stream of data generated by the mappers correctly.

If the mappers output exactly the same records as they receive, then you can omit this option.

map.split
The number of rows that your mapper function receives from a mapper.

- **0** sends all rows given by Hadoop to a specific mapper to your mapper function. Use this setting only if you are sure that the chunk of data for each mapper can fit into R memory. If it does not, then the R process will fail with a memory allocation error.
- **1** sends one row only to the mapper at a time (Default). You can improve the performance of a mapper by increasing this value, which decreases the number of invocations of your mapper function. This is particularly important for small functions.
- ***n*** sends a minimum of *n* rows to the mapper at a time. In this syntax, *n* is an integer greater than 1. Some algorithms require a minimum number of rows to function.

map.tasks
The number of mappers to run. Specify `1` to run the mappers sequentially; specify a larger integer to run the mappers in parallel. If you do not specify a value, then the number of mappers is determined by the capacity of the cluster, the workload, and the Hadoop configuration.

map.valkey
Set to `TRUE` to duplicate the keys as data values for the mapper, or `FALSE` to use the keys only as keys (default).

min.split.size
Controls the lower boundary for splitting HDFS files before sending them to the mappers. This option reflects the value of the Hadoop `mapred.min.split.size` option and is set in bytes.

reduce.input
A reducer input data type keyword: `data.frame` or `list` (default).

reduce.output
A sample R data frame object that defines the output structure of data from the reducers. This is optional parameter.

The sample data frame is used to generate metadata for the output HDFS objects. It reduces the job execution time by eliminating the sampling and parsing step. It also results in a better output data format, because you can specify column names and other metadata. If you do not specify a sample data frame, then the output HDFS files are sampled and the metadata is generated automatically.

reduce.split
The number of data values given at one time to the reducer. See the values for `map.split`. The reducer expects to receive all values at one time, so you must handle partial data sets in the reducer if you set `reduce.split` to a value other than 0 (zero).

reduce.tasks
The number of reducers to run. Specify `1` to run the reducers sequentially; specify a larger integer to run the reducers in parallel. If you do not specify a value, then the number of reducers is determined by the capacity of the cluster, the workload, and the Hadoop configuration.

reduce.valkey
Set to `TRUE` to duplicate the keys as data values for the reducer, or `FALSE` to use the keys only as keys (default).

verbose
Set to `TRUE` to generate diagnostic information, or `FALSE` otherwise.

Examples and Demos of Oracle R Connector for Hadoop

The `ORCH` package includes sample code to help you learn to adapt your R programs to run on a Hadoop cluster using Oracle R Connector for Hadoop. This topic describes these examples and demonstrations.

- Using the Demos
- Using the Examples

Using the Demos

Oracle R Connector for Hadoop provides an extensive set of demos. Instructions for running them are included in the following descriptions.

If an error occurs, then exit from R without saving the workspace image and start a new session. You should also delete the temporary files created in both the local file system and the HDFS file system:

```
# rm -r /tmp/orch*
# hadoop fs -rm -r /tmp/orch*
```

Demo R Programs

hdfs_cpmv.R
Demonstrates copying, moving, and deleting HDFS files and directories.

This program uses the following `ORCH` functions:

`hdfs.cd`
`hdfs.cp`
`hdfs.exists`
`hdfs.ls`
`hdfs.mkdir`
`hdfs.mv`

hdfs.put
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot

To run this demo, use the following command:

```
R> demo("hdfs_cpmv", package="ORCH")
```

hdfs_datatrans.R
Demonstrates data transfers between HDFS and the local file system, and between HDFS and an Oracle database.

> **Note:** This demo requires that Oracle R Enterprise client is installed on Oracle Big Data Appliance, and Oracle R Enterprise server is installed on the Oracle Database host.
>
> You must connect to Oracle Database using both Oracle R Connector for Hadoop and Oracle R Enterprise to run this demo to completion. See orch.connect on page 6-41 and ore.connect help.

This program uses the following ORCH functions:

hdfs.cd
hdfs.describe
hdfs.download
hdfs.get
hdfs.pull
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.setroot
hdfs.size
hdfs.upload
orch.connected
ore.drop[1]
ore.is.connected[1]

To run this demo, use the following commands, replacing the parameters shown here for ore.connect and orch.connect with those appropriate for your Oracle database:

```
R> ore.connect("RQUSER", "orcl", "localhost", "welcome1")
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("localhost", "RQUSER", "orcl", "welcome1", secure=F)
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID:  orcl
    User: RQUSER
Connected.
[1] TRUE
R> demo("hdfs_datatrans", package="ORCH")
```

hdfs_dir.R
Demonstrates the use of functions related to HDFS directories.

[1] Use the R help function for usage information

This program uses the following ORCH functions:

```
hdfs.cd
hdfs.ls
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot
```

To run this demo, use the following command:

```
R> demo("hdfs_dir", package="ORCH")
```

hdfs_putget.R
Demonstrates how to transfer data between an R data frame and HDFS.

This program uses the following ORCH functions:

```
hdfs.attach
hdfs.describe
hdfs.exists
hdfs.get
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.sample
hdfs.setroot
```

To run this demo, use the following command:

```
R> demo("hdfs_putget", package="ORCH")
```

hive_aggregate.R
Moves a selection of the Iris data set into Hive and performs these aggregations for each species: summary, mean, minimum, maximum, standard deviation, median, and interquartile range (IQR).

This program uses the following ORCH functions to set up a Hive table. These functions are used in all of the Hive demo programs and are documented in R Help:

```
ore.attach
ore.connect
ore.push
```

To run this demo, use the following command:

```
R> demo("hive_aggregate", package="ORCH")
```

hive_analysis.R
Demonstrates basic analysis and data processing operations.

To run this demo, use the following command:

```
R> demo("hive_analysis", package="ORCH")
```

hive_basic.R
Using basic R functions, this demo obtains information about a Hive table, including the number of rows (nrow), the number of columns (length), the first five rows (head), the class and data type of a column (class and is.*), and the number of characters in each column value (nchar).

To run this program, use the following command:

```
R> demo("hive_basic", package="ORCH")
```

hive_binning.R
Creates bins in Hive based on petal size in the Iris data set.

To run this program, use the following command:

```
R> demo("hive_binning", package="ORCH")
```

hive_columnfns.R
Uses the Iris data set to show the use of several column functions including `min`, `max`, `sd`, `mean`, `fivenum`, `var`, `IQR`, `quantile`, `log`, `log2`, `log10`, `abs`, and `sqrt`.

To run this program, use the following command:

```
R> demo("hive_columnfns", package="ORCH")
```

hive_nulls.R
Demonstrates the differences between handling nulls in R and Hive, using the `AIRQUALITY` data set.

This program uses the following `ORCH` functions, which are documented in R Help:

```
ore.attach
ore.connect
ore.na.extract
ore.push
```

To run this program, use the following command:

```
R> demo("hive_nulls", package="ORCH")
```

hive_pushpull.R
Shows processing of the Iris data set split between Hive and the client R desktop.

This program uses the following `ORCH` functions, which are documented in R Help:

```
ore.attach
ore.connect
ore.pull
ore.push
```

To run this program, use the following command:

```
R> demo("hive_pushpull", package="ORCH")
```

hive_sequencefile.R
Shows how to create and use Hive tables stored as sequence files, using the `cars` data set.

This program uses the following `ORCH` functions, which are documented in R Help:

```
ore.attach
ore.connect
ore.create
ore.exec
ore.sync
ore.drop
```

To run this program, use the following command:

```
R> demo("hive_sequencefile", package="ORCH")
```

mapred_basic.R
Provides a simple example of mappers and reducers written in R. It uses the `cars` data set.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
hdfs.rm
hdfs.setroot
orch.keyval
```

To run this program, use the following command:

```
R> demo("mapred_basic", package="ORCH")
```

mapred_modelbuild.R
Runs a mapper and a reducer in a parallel model build, saves plots in HDFS in parallel, and displays the graphs. It uses the `iris` data set.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.download
hdfs.exists
hdfs.get
hdfs.id
hdfs.put
hdfs.rm
hdfs.rmdir
hdfs.root
hdfs.setroot
hdfs.upload
orch.export
orch.pack
orch.unpack
```

To run this program, use the following command:

```
R> demo("mapred_modelbuild", package="ORCH")
```

orch_lm.R
Fits a linear model using the `orch.lm` algorithm and compares it with the R `lm` algorithm. It uses the `iris` data set.

This program uses the following `ORCH` functions:

```
hdfs.cd
hdfs.get
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.setroot
orch.lm[1]
predict.orch.lm[1]
```

To run this program, use the following command:

```
R> demo("orch_lm", package="ORCH")
```

orch_lmf_jellyfish.R
Fits a low rank matrix factorization model using the jellyfish algorithm.

This program uses the following ORCH functions:

hdfs.cd
hdfs.get
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot
orch.evaluate[1]
orch.export.fit[1]
orch.lmf[1]

To run this program, use the following command:

```
R> demo("orch_lmf_jellyfish", package="ORCH")
```

orch_lmf_mahout_als.R
Fits a low rank matrix factorization model using the Mahout ALS-WR algorithm.

This program uses the following ORCH functions:

hdfs.cd
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
orch.evaluate[1]
orch.lmf[1]
orch.recommend[1]

To run this program, use the following command:

```
R> demo("orch_lmf_mahout_als", package="ORCH")
```

orch_neural.R
Builds a model using the Oracle R Connector for Hadoop neural network algorithm. It uses the iris data set.

This program uses the following ORCH functions:

hdfs.cd
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.sample
hdfs.setroot
orch.neural[1]

To run this program, use the following command:

```
R> demo("orch_neural", package="ORCH")
```

orch_nmf.R
Demonstrates the integration of the orch.nmf function with the framework of the R NMF package to create a nonnegative matrix factorization model.

This demo uses the Golub data set provided with the NMF package.

This program uses the following ORCH functions:

hdfs.cd
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot
orch.nmf.NMFalgo[1]

This program requires several R packages that you may not have already installed: NMF, Biobase, BiocGenerics, and RColorBrewer. Instructions for installing them are provided here.

> **Caution:** When asked whether you want to update the dependent packages, always respond **Update/None**. The installed R packages are the supported versions. If you update them, then the Oracle Big Data Appliance software checks fail, which causes problems during routine maintenance. Always run the bdachecksw utility after installing new packages. See the bdachecksw utility in the *Oracle Big Data Appliance Owner's Guide*.
>
> If you are using a Hadoop client, then install the packages on the client instead of Oracle Big Data Appliance.

To install NMF:

1. Download NMF from the CRAN archives at

   http://cran.r-project.org/src/contrib/Archive/NMF/NMF_0.5.06.tar.gz

2. Install NMF using a standard installation method:

   ```
   R> install.packages("/full_path/NMF_0.5.06.tar.gz", REPOS=null)
   ```

To install Biobase and BiocGenerics from the BioConductor project:

1. Source in the biocLite package:

   ```
   R> source("http://bioconductor.org/biocLite.R")
   ```

2. Install the Biobase package:

   ```
   R> biocLite("Biobase")
   ```

3. Install the BiocGenerics package:

   ```
   R> biocLite("BiocGenerics")
   ```

To install RColorBrewer:

- Use a standard method of installing a package from CRAN:

  ```
  R> install.packages("RColorBrewer")
  ```

To run the orch_nmf demo:

1. Load the required packages, if they are not already loaded in this session:

   ```
   R> library("NMF")
   R> library("Biobase")
   R> library("BiocGenerics")
   R> library("RColorBrewer")
   ```

2. Run the demo:

```
R> demo("orch_nmf", package="ORCH")
```

demo-bagged.clust.R
Provides an example of bagged clustering using randomly generated values. The mappers perform k-means clustering analysis on a subset of the data and generate centroids for the reducers. The reducers combine the centroids into a hierarchical cluster and store the `hclust` object in HDFS.

This program uses the following `ORCH` functions:

```
hadoop.exec
hdfs.put
is.hdfs.id
orch.export
orch.keyval
orch.pack
orch.unpack
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/demos/demo-bagged.clust.R")

Call:
c("hclust", "d", "single")

Cluster method   : single
Distance         : euclidean
Number of objects: 6
```

demo-kmeans.R
Performs k-means clustering.

This program uses the following `ORCH` functions:

```
hadoop.exec
hdfs.get
orch.export
orch.keyval
orch.pack
orch.unpack
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/demos/demo-kmeans.R")
centers
           x          y
1  0.3094056  1.32792762
2  1.7374920 -0.06730981
3 -0.6271576 -1.20558920
4  0.2668979 -0.98279426
5  0.4961453  1.11632868
6 -1.7328809 -1.26335598
removing NaNs if any
final centroids
             x          y
1  0.666782828  2.36620810
2  1.658441962  0.33922811
```

```
3 -0.627157587 -1.20558920
4 -0.007995672 -0.03166983
5  1.334578589  1.41213532
6 -1.732880933 -1.26335598
```

Using the Examples

The examples show how you use the Oracle R Connector for Hadoop API. You can view them in a text editor after extracting them from the installation archive files. They are located in the ORCH2.1.0/ORCHcore/examples directory. You can run the examples as shown in the following descriptions.

Example R Programs

example-filter1.R

Shows how to use key-value pairs. The mapper function selects cars with a distance value greater than 30 from the `cars` data set, and the reducer function calculates the mean distance for each speed.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter1.R")
Running cars filtering and mean example #1:
   val1     val2
1    10 34.00000
2    13 38.00000
3    14 58.66667
4    15 54.00000
5    16 36.00000
6    17 40.66667
7    18 64.50000
8    19 50.00000
9    20 50.40000
10   22 66.00000
11   23 54.00000
12   24 93.75000
13   25 85.00000
```

example-filter2.R

Shows how to use values only. The mapper function selects cars with a distance greater than 30 and a speed greater than 14 from the `cars` data set, and the reducer function calculates the mean speed and distance as one value pair.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter2.R")
Running cars filtering and mean example #2:
   val1  val2
1 59.52 19.72
```

example-filter3.R

Shows how to load a local file into HDFS. The mapper and reducer functions are the same as example-filter2.R.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.download
hdfs.upload
orch.keyval
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter3.R")
Running cars filtering and mean example #3:
[1] "\t59.52,19.72"
```

example-group.apply.R

Shows how to build a parallel model and generate a graph. The mapper partitions the data based on the petal lengths in the iris data set, and the reducer uses basic R statistics and graphics functions to fit the data into a linear model and plot a graph.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.download
hdfs.exists
hdfs.get
hdfs.id
hdfs.mkdir
hdfs.put
hdfs.rmdir
hdfs.upload
orch.pack
orch.export
orch.unpack
```

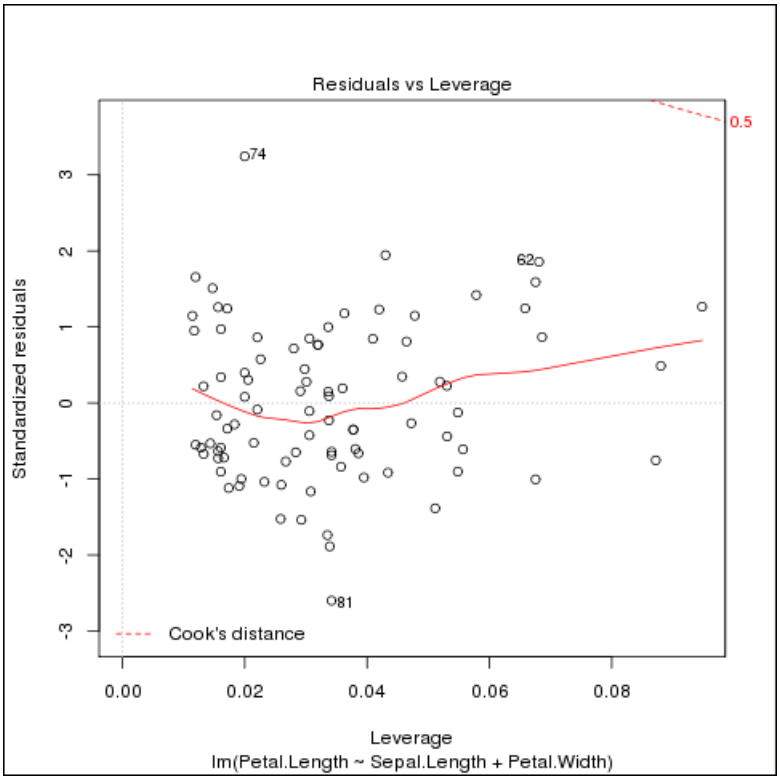
To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-group.apply.R")
Running groupapply example.
[[1]]
[[1]]$predict
     .
     .
     .
[3]
[[3]]$predict
       2      210        3        4        5        6        7        8
4.883662 5.130816 4.854156 5.163097 4.109921 4.915944 4.233498 4.886437
       9       10       11       12       13       14       15       16
```

```
4.789593 4.545214 4.136653 4.574721 4.945451 4.359849 4.013077 4.730579
      17       18       19       20       21       22       23       24
4.827423 4.480651 4.792367 4.386581 4.666016 5.007239 5.227660 4.607002
      25       26       27       28       29       30       31       32
4.701072 4.236272 4.701072 5.039521 4.604228 4.171709 4.109921 4.015851
      33       34       35       36       37       38       39       40
4.574721 4.201216 4.171709 4.139428 4.233498 4.542439 4.233498 5.733065
      41       42       43       44       45       46       47       48
4.859705 5.851093 5.074577 5.574432 6.160034 4.115470 5.692460 5.321730
      49       50       51       52       53       54       55       56
6.289160 5.386293 5.230435 5.665728 4.891986 5.330054 5.606714 5.198153
      57       58       59       60       61       62       63       64
6.315892 6.409962 4.607002 5.915656 4.830198 6.127753 5.074577 5.603939
      65       66       67       68       69       70       71       72
5.630671 5.012788 4.951000 5.418574 5.442532 5.848318 6.251329 5.512644
      73       74       75       76       77       78       79       80
4.792367 4.574721 6.409962 5.638995 5.136365 4.889212 5.727516 5.886149
      81       82       83       84       85       86       87       88
5.915656 4.859705 5.853867 5.980218 5.792079 5.168646 5.386293 5.483137
      89
4.827423

[[3]]$pngfile
[1] "/user/oracle/pngfiles/3"

[1] "/tmp/orch6e295a5a5da9"
```

This program generates three graphics files in the /tmp directory. Figure 5–1 shows the last one.

Figure 5–1 Example example-group.apply.R Output in fit-3.png



example-kmeans.R
Defines a k-means clustering function and generates random points for a clustering test. The results are printed or graphed.

This program uses the following ORCH functions:

```
hadoop.exec
hdfs.get
hdfs.put
orch.export
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-kmeans.R")
Running k-means example.
         val1      val2
1 1.005255389 1.9247858
2 0.008390976 2.5178661
3 1.999845464 0.4918541
4 0.480725254 0.4872837
5 1.677254045 2.6600670
```

example-lm.R
Shows how to define multiple mappers and one reducer that merges all results. The program calculates a linear regression using the iris data set.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.export
orch.pack
orch.unpack
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-lm.R")
Running linear model example.
Model rank 3, yy 3.0233000000E+02, nRows 150
Model coefficients
-0.2456051 0.2040508 0.5355216
```

example-logreg.R
Performs a one-dimensional, logistic regression on the cars data set.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-logreg.R")
Running logistic regression.
[1] 1924.1
```

example-map.df.R

Shows how to run the mapper with an unlimited number of records at one time input as a data frame. The mapper selects cars with a distance greater than 30 from the `cars` data set and calculates the mean distance. The reducer merges the results.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-map.df.R")
Running example of data.frame mapper input:
      val1     val2
1 17.66667 50.16667
2 13.25000 47.25000
```

example-map.list.R

Shows how to run the mapper with an unlimited number of records at one time input as a list. The mapper selects cars with a distance greater than 30 from the `cars` data set and calculates the mean distance. The reducer merges the results.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-map.list.R"
Running example of list mapper input:
  val1 val2
1 15.9   49
```

example-model.plot.R

Shows how to create models and graphs using HDFS. The mapper provides key-value pairs from the `iris` data set to the reducer. The reducer creates a linear model from data extracted from the data set, plots the results, and saves them in three HDFS files.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.download
hdfs.exists
hdfs.get
hdfs.id
hdfs.mkdir
hdfs.put
hdfs.rmdir
hdfs.upload
orch.export
orch.pack
```

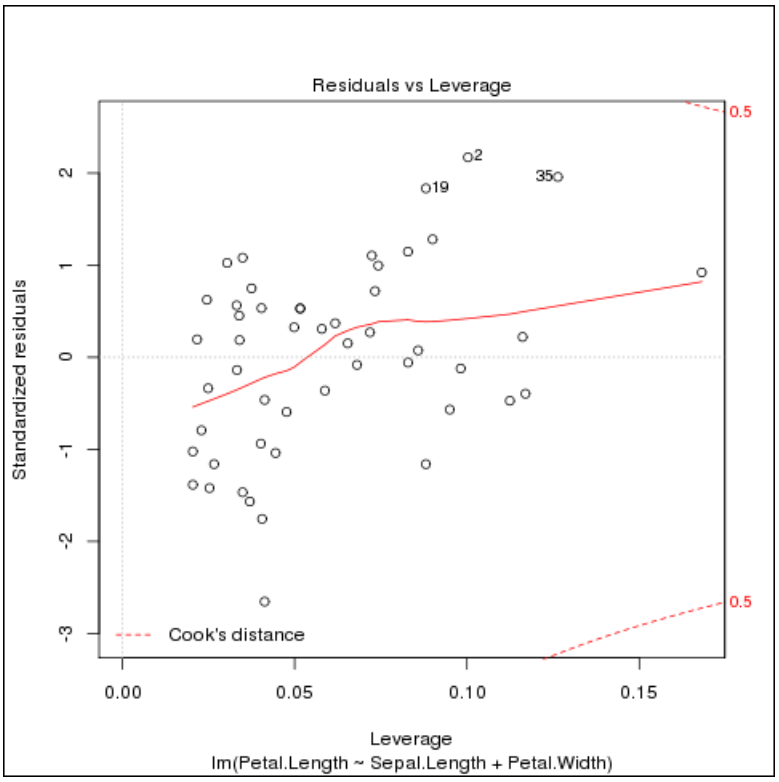
To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
```

```
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-model.plot.R"
Running model building and graph example.
[[1]]
[[1]]$predict
     .
     .
     .
[[3]]$predict
       2      210        3        4        5        6        7        8
5.423893 4.955698 5.938086 5.302979 5.517894 6.302640 4.264954 6.032087
       9       10       11       12       13       14       15       16
5.594622 6.080090 5.483347 5.393163 5.719353 4.900061 5.042065 5.462257
      17       18       19       20       21       22       23       24
5.448801 6.392824 6.410097 5.032427 5.826811 4.827150 6.358277 5.302979
      25       26       27       28       29       30       31       32
5.646442 5.959176 5.230068 5.157157 5.427710 5.924630 6.122271 6.504099
      33       34       35       36       37       38       39       40
5.444983 5.251159 5.088064 6.410097 5.406619 5.375890 5.084247 5.792264
      41       42       43       44       45       46       47       48
5.698262 5.826811 4.955698 5.753900 5.715536 5.680989 5.320252 5.483347
      49       50
5.316435 5.011336

[[3]]$pngfile
[1] "/user/oracle/pngfiles/virginica"


[1] "/tmp/orch6e29190de160"
```

The program generates three graphics files. Figure 5–2 shows the last one.

Figure 5–2 Example example-model.plot.R Output in fit-virginica.png



example-model.prep.R
Shows how to distribute data across several map tasks. The mapper generates a data frame from a slice of input data from the `iris` data set. The reducer merges the data frames into one output data set.

This program uses the following `ORCH` functions:

```
hadoop.exec
hdfs.get
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-model.prep.R")
     v1    v2          v3          v4
1   5.1  4.90    4.055200 0.33647224
2   4.9  4.20    4.055200 0.33647224
3   4.7  4.16    3.669297 0.26236426
      .
      .
      .
299 6.2 18.36 221.406416 1.68639895
300 5.9 15.30 164.021907 1.62924054
```

example-rlm.R
Shows how to convert a simple R program into one that can run as a MapReduce job on a Hadoop cluster. In this example, the program calculates and graphs a linear model on the `cars` data set using basic R functions.

This program uses the following ORCH functions:

hadoop.run
orch.keyvals
orch.unpack

To run this program, ensure that root is set to /tmp, and then source the file:

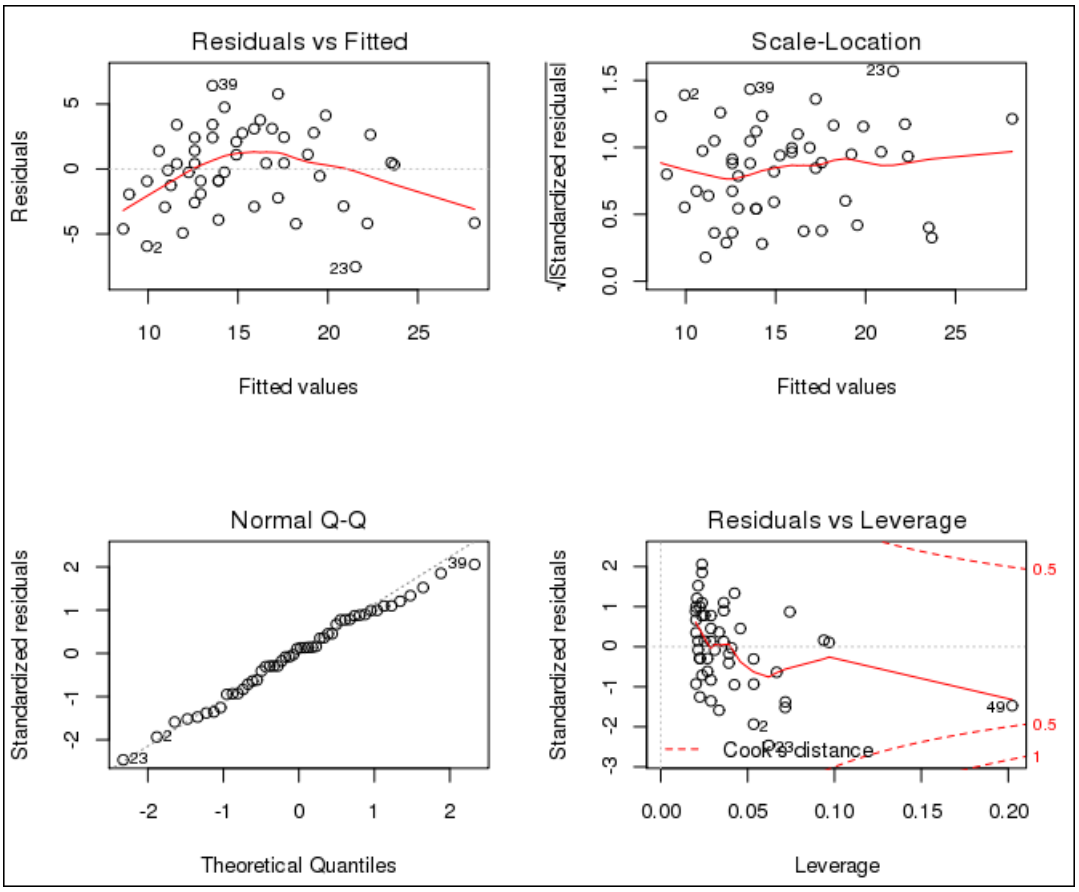
```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-rlm.R"
[1] "--- Client lm:"

Call:
lm(formula = speed ~ dist, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max
-7.5293 -2.1550  0.3615  2.4377  6.4179

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
dist         0.16557    0.01749   9.464 1.49e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.156 on 48 degrees of freedom
Multiple R-squared: 0.6511,     Adjusted R-squared: 0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Enter Ctrl+C to get the R prompt, and then close the graphics window:

```
R> dev.off()
```

Figure 5–3 shows the four graphs generated by the program.

Figure 5–3 Example-rlm.R Output



example-split.map.R
Shows how to split the data in the mapper. The first job runs the mapper in list mode and splits the list in the mapper. The second job splits a data frame in the mapper. Both jobs use the `cars` data set.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-split.map.R")
Running example of list splitting in mapper
  key count splits
1  NA    50      6
Running example of data.frame splitting in mapper
  key count splits
1  NA    50      8
```

example-split.reduce.R
Shows how to split the data from the `cars` data set in the reducer.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-split.reduce.R")
Running example of data.frame reducer input
^C   key count
1    1     9
2    1     9
3    1     9
    .
    .
    .
20   1     9
21   1     5
```

example-sum.R

Shows how to perform a sum operation in a MapReduce job. The first job sums a vector of numeric values, and the second job sums all columns of a data frame.

This program uses the following `ORCH` functions:

```
hadoop.run
orch.keyval
```

To run this program, ensure that `root` is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-sum.R"
  val2
1    6
  val2 val3
1  770 2149
```

example-teragen.matrix.R

Shows how to generate large data sets in a matrix for testing programs in Hadoop. The mappers generate samples of random data, and the reducers merge them.

This program uses the following `ORCH` functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that `root` is set to /tmp, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragem.matrix.R"
Running TeraGen-PCA example:
R>
```

example-teragen.xy.R
Shows how to generate large data sets in a data frame for testing programs in Hadoop. The mappers generate samples of random data, and the reducers merge them.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that root is set to /tmp, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragen2.xy.R")
Running TeraGen2 example.R>
```

example-teragen2.xy.R
Shows how to generate large data sets in a data frame for testing programs in Hadoop. One mapper generates small samples of random data, and the reducers merge them.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that root is set to /tmp, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragem2.xy.R"
Running TeraGen2 example.
R>
```

example-terasort.R
Provides an example of a TeraSort job on a set of randomly generated values.

This program uses the following ORCH function:

```
hadoop.run
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-terasort.R"
Running TeraSort example:
          val1
1 -0.001467344
2 -0.004471376
3 -0.005928546
4 -0.007001193
5 -0.010587280
6 -0.011636190
```

Security Notes for Oracle R Connector for Hadoop

Oracle R Connector for Hadoop can invoke the Sqoop utility to connect to Oracle Database either to extract data or to store results.

Sqoop is a command-line utility for Hadoop that imports and exports data between HDFS or Hive and structured databases. The name Sqoop comes from "SQL to Hadoop." The following explains how Oracle R Connector for Hadoop stores a database user password and sends it to Sqoop.

Oracle R Connector for Hadoop stores a user password only when the user establishes the database connection in a mode that does not require reentering the password each time. The password is stored encrypted in memory. See `orch.connect` on page 6-41.

Oracle R Connector for Hadoop generates a configuration file for Sqoop and uses it to invoke Sqoop locally. The file contains the user's database password obtained by either prompting the user or from the encrypted in-memory representation. The file has local user access permissions only. The file is created, the permissions are set explicitly, and then the file is open for writing and filled with data.

Sqoop uses the configuration file to generate custom JAR files dynamically for the specific database job and passes the JAR files to the Hadoop client software. The password is stored inside the compiled JAR file; it is not stored in plain text.

The JAR file is transferred to the Hadoop cluster over a network connection. The network connection and the transfer protocol are specific to Hadoop, such as port 5900.

The configuration file is deleted after Sqoop finishes compiling its JAR files and starts its own Hadoop jobs.

6

ORCH Library Reference

This chapter describes the function in the ORCH library. It contains the following sections:

- Functions in Alphabetical Order
- Functions by Category

Functions in Alphabetical Order

Use R help for the `ore` functions for Hive and the analytic functions:

```
R> help("function_name")
```

There are no Help (Rd) files for `orch` functions. They are described in this chapter.

```
as.ore
as.ore.character
as.ore.frame
as.ore.integer
as.ore.logical
as.ore.numeric
as.ore.vector
hadoop.exec
hadoop.run
hdfs.attach
hdfs.cd
hdfs.cp
hdfs.describe
hdfs.download
hdfs.exists
hdfs.get
hdfs.head
hdfs.id
hdfs.ls
hdfs.mkdir
hdfs.mv
hdfs.parts
hdfs.pull
hdfs.push
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.rmdir
hdfs.root
hdfs.sample
hdfs.setroot
```

```
hdfs.size
hdfs.tail
hdfs.upload
is.hdfs.id
is.ore
is.ore.character
is.ore.frame
is.ore.integer
is.ore.logical
is.ore.numeric
is.ore.vector
ore.create
ore.drop
ore.get
ore.pull
ore.push
ore.recode
orch.connect
orch.connected
orch.dbcon
orch.dbg.lasterr
orch.dbg.off
orch.dbg.on
orch.dbg.output
orch.dbinfo
orch.disconnect
orch.dryrun
orch.evaluate
orch.export
orch.export.fit
orch.keyval
orch.keyvals
orch.lm
orch.lmf
orch.neural
orch.nmf
orch.nmf.NMFalgo
orch.pack
orch.reconnect
orch.temp.path
orch.unpack
orch.version
predict.orch.lm
print.orch.lm
print.summary.orch.lm
summary.orch.lm
```

Functions by Category

The functions are grouped into these categories:

- Making Connections
- Copying Data
- Exploring Files
- Writing MapReduce Functions
- Debugging Scripts
- Using Hive Data

- Writing Analytical Functions

Making Connections

orch.connect
orch.connected
orch.dbcon
orch.dbinfo
orch.disconnect
orch.reconnect

Copying Data

hdfs.attach
hdfs.cp
hdfs.download
hdfs.get
hdfs.mv
hdfs.pull
hdfs.push
hdfs.put
hdfs.upload
orch.export
orch.pack
orch.unpack

Exploring Files

hdfs.cd
hdfs.describe
hdfs.exists
hdfs.head
hdfs.id
hdfs.ls
hdfs.mkdir
hdfs.parts
hdfs.pwd
hdfs.rm
hdfs.rmdir
hdfs.root
hdfs.sample
hdfs.setroot
hdfs.size
hdfs.tail
is.hdfs.id
orch.temp.path

Writing MapReduce Functions

hadoop.exec
hadoop.run
orch.dryrun
orch.keyval
orch.keyvals

Debugging Scripts

orch.dbg.lasterr
orch.dbg.off

```
orch.dbg.on
orch.dbg.output
orch.version
```

Using Hive Data

See "ORE Functions for Hive" on page 5-3

Writing Analytical Functions

See "Analytic Functions in Oracle R Connector for Hadoop" on page 5-9.

hadoop.exec

Starts the Hadoop engine and sends the mapper, reducer, and combiner R functions for execution. You must load the data into HDFS first.

Usage

```
hadoop.exec(
        dfs.id,
        mapper,
        reducer,
        combiner,
        export,
        init,
        final,
        job.name,
        config)
```

Arguments

dfs.id
The name of a file in HDFS containing data to be processed. The file name can include a path that is either absolute or relative to the current path.

mapper
Name of a mapper function written in the R language.

reducer
Name of a reducer function written in the R language (optional).

combiner
Not supported in this release.

export
Names of exported R objects from your current R environment that are referenced by any of the mapper, reducer, or combiner functions (optional).

init
A function that is executed once before the mapper function begins (optional).

final
A function that is executed once after the reducer function completes (optional).

job.name
A descriptive name that you can use to track the progress of the MapReduce job instead of the automatically generated job name (optional).

config
Sets the configuration parameters for the MapReduce job (optional).

This argument is an instance of the `mapred.config` class, and thus it has this format:

```
config = new("mapred.config", param1, param2,...)
```

See "ORCH mapred.config Class" on page 5-9 for a description of this class.

Usage Notes

Oracle R Connector for Hadoop does not support mapper-only jobs. Use orch.keyvals as a reducer body. See the example in `orch.keyvals`.

This function provides more control of the data flow than the `hadoop.run` function. You must use `hadoop.exec` when chaining several mappers and reducers in a pipeline, because the data does not leave HDFS. The results are stored in HDFS files.

Return Value

Data object identifier in HDFS

See Also

`hadoop.run` on page 6-8, `orch.dryrun` on page 6-54, orch.keyvals on page 6-57

Example

This sample script uses `hdfs.attach` to obtain the object identifier of a small, sample data file in HDFS named ontime_R.

The MapReduce function counts the number of on-time flights arriving in the San Francisco International Airport (SFO).

```
dfs <- hdfs.attach('ontime_R')
res <- NULL
res <- hadoop.exec(
    dfs,
    mapper = function(key, ontime) {
        if (key == 'SFO') {
            keyval(key, ontime)
        }
    },
    reducer = function(key, vals) {
        sumAD <- 0
        count <- 0
        for (x in vals) {
           if (!is.na(x$ARRDELAY)) {sumAD <- sumAD + x$ARRDELAY; count <- count +
1}
        }
        res <- sumAD / count
        keyval(key, res)
    }
)
```

After the script runs, the `res` variable identifies the location of the results in an HDFS file named /user/oracle/xq/orch3d0b8218:

```
R> res
[1] "/user/oracle/xq/orch3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
 val1     val2
1 SFO 27.05804
```

This code fragment is extracted from example-kmeans.R. The export option identifies the location of the `ncenters` generated data set, which is exported as an HDFS file. The `config` options provide a MapReduce job name of `k-means.1`, and the mapper output format of a data frame.

```
mapf <- data.frame(key=0, val1=0, val2=0)
   dfs.points <- hdfs.put(points)
   dfs.centers <- hadoop.exec(
       dfs.id = dfs.points,
       mapper = function(k,v) {
           keyval(sample(1:ncenters,1), v)
       },
       reducer = function(k,vv) {
           vv <- sapply(vv, unlist)
           keyval(NULL, c(mean(vv[1,]), mean(vv[2,])))
       },
       export = orch.export(ncenters),
       config = new("mapred.config",
           job.name = "k-means.1",
           map.output = mapf)
```

hadoop.run

Starts the Hadoop engine and sends the mapper, reducer, and combiner R functions for execution. If the data is not already stored in HDFS, then `hadoop.run` first copies the data there.

Usage

```
hadoop.run(
        data,
        mapper,
        reducer,
        combiner,
        export,
        init,
        final,
        job.name,
        config)
```

Arguments

data
A data frame, Oracle R Enterprise frame (`ore.frame`), or an HDFS file name.

mapper
The name of a mapper function written in the R language.

reducer
The name of a reducer function written in the R language (optional).

combiner
Not supported in this release.

export
The names of exported R objects.

init
A function that is executed once before the mapper function begins (optional).

final
A function that is executed once after the reducer function completes (optional).

job.name
A descriptive name that you can use to track the progress of the job instead of the automatically generated job name (optional).

config
Sets the configuration parameters for the MapReduce job (optional).

This argument is an instance of the `mapred.config` class, and so it has this format:

```
config = new("mapred.config", param1, param2,...
```

See "ORCH mapred.config Class" on page 5-9 for a description of this class.

Usage Notes

Oracle R Connector for Hadoop does not support mapper-only jobs. Use orch.keyvals as a reducer body. See the example in `orch.keyvals`.

The `hadoop.run` function returns the results from HDFS to the source of the input data. For example, the results for HDFS input data are kept in HDFS, and the results for `ore.frame` input data are copied into an Oracle database.

Return Value

An object in the same format as the input data

See Also

`hadoop.exec` on page 6-5, `orch.dryrun` on page 6-54, orch.keyvals on page 6-57

Example

This sample script uses `hdfs.attach` to obtain the object identifier of a small, sample data file in HDFS named ontime_R.

The MapReduce function counts the number of on-time flights arriving in the San Francisco International Airport (SFO).

```
dfs <- hdfs.attach('ontime_R')
res <- NULL
res <- hadoop.run(
    dfs,
    mapper = function(key, ontime) {
        if (key == 'SFO') {
            keyval(key, ontime)
        }
    },
    reducer = function(key, vals) {
        sumAD <- 0
        count <- 0
        for (x in vals) {
           if (!is.na(x$ARRDELAY)) {sumAD <- sumAD + x$ARRDELAY; count <- count +
1}
        }
        res <- sumAD / count
        keyval(key, res)
    }
)
```

After the script runs, the location of the results is identified by the `res` variable, in an HDFS file named /user/oracle/xq/orch3d0b8218:

```
R> res
[1] "/user/oracle/xq/orch3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
 val1     val2
1 SFO 27.05804
```

The next example shows o

```
hadoop.run(x,
    mapper = function(k,v) {
        orch.keyval(k, v+1) # increment all values
    },
```

```
reducer = function(k, vv) {
    orch.keyvals(k, vv) # pass-through
}
```

hdfs.attach

Copies data from an unstructured data file in HDFS into the Oracle R Connector for Hadoop framework. By default, data files in HDFS are not visible to the connector. However, if you know the name of the data file, you can use this function to attach it to the Oracle R Connector for Hadoop name space.

Usage

```
hdfs.attach(
        dfs.name,
        force)
```

Arguments

dfs.name
The name of a file in HDFS.

force
Controls whether the function attempts to discover the structure of the file and the data type of each column.

`FALSE` for comma-separated value (CSV) files (default). If a file does not have metadata identifying the names and data types of the columns, then the function samples the data to deduce the data type as number or string. It then re-creates the file with the appropriate metadata.

`TRUE` for non-CVS files, including binary files. This setting prevents the function from trying to discover the metadata; instead, it simply attaches the file.

Usage Notes

Use this function to attach a CSV file to your R environment, just as you might attach a data frame.

Oracle R Connector for Hadoop does not support the processing of attached non-CVS files. Nonetheless, you can attach a non-CSV file, download it to your local computer, and use it as desired. Alternatively, you can attach the file for use as input to a Hadoop application.

This function can become slow when processing large input HDFS files, as the result of inherited limitations in the Hadoop command-line interface.

Return Value

The object ID of the file in HDFS, or `NULL` if the operation failed

See Also

`hdfs.download` on page 6-16

Example

This example stores the object ID of `ontime_R` in a variable named `dfs`, and then displays its value.

```
R> dfs <- hdfs.attach('ontime_R')
R> dfs
```

```
[1] "/user/oracle/xq/ontime_R"
attr(,"dfs.id")
[1] TRUE
```

hdfs.cd

Sets the default HDFS path.

Usage

```
hdfs.cd(dfs.path)
```

Arguments

dfs.path
A path that is either absolute or relative to the current path.

Return Value

`TRUE` if the path is changed successfully, or `FALSE` if the operation failed

Example

This example changes the current directory from /user/oracle to /user/oracle/sample:

```
R> hdfs.cd("sample")
[1] "/user/oracle/sample"
```

hdfs.cp

Copies an HDFS file from one location to another.

Usage

```
hdfs.cp(
        dfs.src,
        dfs.dst,
        force)
```

Arguments

dfs.src
The name of the source file to be copied. The file name can include a path that is either absolute or relative to the current path.

dfs.dst
The name of the copied file. The file name can include a path that is either absolute or relative to the current path.

force
Set to `TRUE` to overwrite an existing file, or set to `FALSE` to display an error message (default).

Return Value

`NULL` for a successful copy, or `FALSE` for a failed attempt

Example

This example copies a file named weblog in the parent directory and overwrites the existing weblog file:

```
R> hdfs.cp("weblog", "..", force=T)
```

hdfs.describe

Returns the metadata associated with a file in HDFS.

Usage

```
hdfs.describe(
        dfs.id)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

Return Value

A data frame containing the metadata, or `NULL` if no metadata was available in HDFS

Example

This example provides information about an HDFS file named ontime_DB:

```
R> hdfs.describe('ontime_DB')
           name
1          path
2        origin
3         class
4         types
5           dim
6         names
7       has.key
8    key.column
9      null.key
10 has.rownames
11         size
12        parts

1                          values
2                       ontime_DB
3                         unknown
     .
     .
     .
```

hdfs.download

Copies a file from HDFS to the local file system.

Usage

```
hdfs.download(
        dfs.id,
        filename,
        overwrite)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

filename
The name of a file in the local file system where the data is copied.

overwrite
Controls whether the operation can overwrite an existing local file. Set to `TRUE` to overwrite *`filename`*, or `FALSE` to signal an error (default).

Usage Notes

This function provides the fastest and easiest way to copy a file from HDFS. No data transformations occur except merging multiple parts into a single file. The local file has the exact same data as the HDFS file.

Return Value

Local file name, or `NULL` if the copy failed

Example

This example displays a list of files in the current HDFS directory and copies ontime2000.DB to the local file system as /home/oracle/ontime2000.dat.

```
R> hdfs.ls()
[1] "ontime2000_DB" "ontime_DB"     "ontime_File"   "ontime_R"      "testdata.dat"
R> tmpfile <- hdfs.download("ontime2000_DB", "/home/oracle/ontime2000.dat",
overwrite=F)
R> tmpfile
[1] "/home/oracle/ontime2000.dat"
```

hdfs.exists

Verifies that a file exists in HDFS.

Usage

```
hdfs.exists(
        dfs.id)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

Usage Notes

If this function returns `TRUE`, then you can attach the data and use it in a `hadoop.run` function. You can also use this function to validate an HDFS identifier and ensure that the data exists.

Return Value

`TRUE` if the identifier is valid and the data exists, or `FALSE` if the object is not found

See Also

`is.hdfs.id` on page 6-40

Example

This example shows that the `ontime_R` file exists.

```
R> hdfs.exists("ontime_R")
[1] TRUE
```

hdfs.get

Copies data from HDFS into a data frame in the local R environment. All metadata is extracted and all attributes, such as column names and data types, are restored if the data originated in an R environment. Otherwise, generic attributes like val1 and val2 are assigned.

Usage

```
hdfs.get(
        dfs.id,
        sep)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

sep
The symbol used to separate fields in the file. A comma (,) is the default separator.

Usage Notes

If the HDFS file is small enough to fit into an in-memory R data frame, then you can copy the file using this function instead of the `hdfs.pull` function. The `hdfs.get` function can be faster, because it does not use Sqoop and thus does not have the overhead incurred by `hdfs.pull`.

Return Value

A `data.frame` object in memory in the local R environment pointing to the exported data set, or `NULL` if the operation failed

Example

This example returns the contents of a data frame named `res`.

```
R> print(hdfs.get(res))
  val1      val2
1   AA 1361.4643
2   AS  515.8000
3   CO 2507.2857
4   DL 1601.6154
5   HP  549.4286
6   NW 2009.7273
7   TW 1906.0000
8   UA 1134.0821
9   US 2387.5000
10  WN  541.1538
```

hdfs.head

Copies a specified number of lines from the beginning of a file in HDFS.

Usage

```
hdfs.head(
        dfs.id,
        n)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

n
The number of lines to retrieve from the file. The default is 10.

Return Value

The first *n* lines of the file.

See Also

hdfs.tail on page 6-37

Example

This example returns the first two lines of ontime_R:

```
R> hdfs.head('ontime_R', 2)
[1]
"\"\",\"YEAR\",\"MONTH\",\"MONTH2\",\"DAYOFMONTH\",\"DAYOFMONTH2\",\"DAYOFWEEK\",\
"DEPTIME\",\"CRSDEPTIME\",\"ARRTIME\",\"CRSARRTIME\",\"UNIQUECARRIER\",\"FLIGHTNUM
\",\"TAILNUM\",\"ACTUALELAPSEDTIME\",\"CRSELAPSEDTIME\",\"AIRTIME\",\"ARRDELAY\",\
"DEPDELAY\",\"ORIGIN\",\"DEST\",\"DISTANCE\",\"TAXIIN\",\"TAXIOUT\",\"CANCELLED\",
\"CANCELLATIONCODE\",\"DIVERTED\""
[2]
"\"1\",1994,8,\"M08\",2,\"D02\",2,840,840,1126,1130,\"NW\",878,NA,106,110,NA,-4,0,
\"MEM\",\"TPA\",656,NA,NA,0,NA,\"0\""
```

hdfs.id

Converts an HDFS path name to an R `dfs.id` object.

Usage

```
hdfs.id(
        dfs.x,
        force)
```

Arguments

dfs.x
A string or text expression that resolves to an HDFS file name.

force
Set to `TRUE` if the file need not exist, or set to `FALSE` to ensure that the file does exist.

Return Value

`TRUE` if the string matches an HDFS file name, or `NULL` if a file by that name is not found

Example

This example creates a `dfs.id` object for /user/oracle/demo:

```
R> hdfs.id('/user/oracle/demo')
[1] "user/oracle/demo"
attr(,"dfs.id")
[1] TRUE
```

The next example creates a `dfs.id` object named `id` for a nonexistent directory named /user/oracle/newdemo, after first failing:

```
R> id<-hdfs.id('/user/oracle/newdemo')
DBG: 16:11:38 [ER] "/user/oracle/newdemo" is not found
R> id<-hdfs.id('/user/oracle/newdemo', force=T)
R> id
[1] "user/oracle/newdemo"
attr(,"dfs.id")
[1] TRUE
```

hdfs.ls

Lists the names of all HDFS directories containing data in the specified path.

Usage

```
hdfs.ls(dfs.path)
```

Arguments

dfs.path
A path relative to the current default path. The default path is the current working directory.

Return Value

A list of data object names in HDFS, or `NULL` if the specified path is invalid

See Also

hdfs.cd on page 6-13

Example

This example lists the subdirectories in the current directory:

```
R> hdfs.ls()
[1] "ontime_DB"   "ontime_FILE"   "ontime_R"
```

The next example lists directories in the parent directory:

```
R> hdfs.ls("..")
[1] "demo"   "input"   "output"   "sample"   "xq"
```

This example returns `NULL` because the specified path is not in HDFS.

```
R> hdfs.ls("/bin")
NULL
```

hdfs.mkdir

Creates a subdirectory in HDFS relative to the current working directory.

Usage

```
hdfs.mkdir(
        dfs.name,
        cd)
```

Arguments

dfs.name
Name of the new directory.

cd
`TRUE` to change the current working directory to the new subdirectory, or `FALSE` to keep the current working directory (default).

Return Value

Full path of the new directory as a string, or `NULL` if the directory was not created

Example

This example creates the `/user/oracle/sample` directory.

```
R> hdfs.mkdir('sample', cd=T)
[1] "/user/oracle/sample"
attr(,"dfs.path")
[1] TRUE
```

hdfs.mv

Moves an HDFS file from one location to another.

Usage

```
hdfs.mv(
        dfs.src,
        dfs.dst,
        force)
```

Arguments

dfs.src
The name of the source file to be moved. The file name can include a path that is either absolute or relative to the current path.

dfs.dst
The name of the moved file. The file name can include a path that is either absolute or relative to the current path.

force
Set to `TRUE` to overwrite an existing destination file, or `FALSE` to cancel the operation and display an error message (default).

Return Value

`NULL` for a successful copy, or `FALSE` for a failed attempt

Example

This example moves a file named weblog to the demo subdirectory and overwrites the existing weblog file:

```
R> hdfs.mv("weblog", "./demo", force=T)
```

hdfs.parts

Returns the number of parts composing a file in HDFS.

Usage

```
hdfs.parts(
        dfs.id)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

Usage Notes

HDFS splits large files into parts, which provide a basis for the parallelization of MapReduce jobs. The more parts an HDFS file has, the more mappers can run in parallel.

Return Value

The number of parts composing the object, or `0` if the object does not exist in HDFS

Example

This example shows that the ontime_R file in HDFS has one part:

```
R> hdfs.parts("ontime_R")
[1] 1
```

hdfs.pull

Copies data from HDFS into an Oracle database.

This operation requires authentication by Oracle Database. See orch.connect on page 6-41.

Usage

```
hdfs.pull(
        dfs.id,
        sep,
        db.name,
        overwrite,
        driver)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

sep
The symbol used to separate fields in the file (optional). A comma (,) is the default separator.

db.name
The name of a table in an Oracle database.

overwrite
Controls whether `db.name` can overwrite a table with the same name. Set to `TRUE` to overwrite the table, or `FALSE` to signal an error (default).

driver
Identifies the driver used to copy the data: `Sqoop` (default) or `olh` to use Oracle Loader for Hadoop. You must set up Oracle Loader for Hadoop before using it as a driver. See the Usage Notes and "Oracle Loader for Hadoop Setup" on page 1-12..

Usage Notes

With the Oracle Advanced Analytics option, you can use Oracle R Enterprise to analyze the data after loading it into an Oracle database.

Choosing a Driver

Sqoop is synchronous, and copying a large data set may take a while. The prompt reappears and you regain use of R when copying is complete.

Oracle Loader for Hadoop is much faster than Sqoop, and so you should use it as the driver if possible.

Correcting Problems With the OLH Driver

If Oracle Loader for Hadoop is available, then you see this message when the `ORCH` library is loading:

```
OLH 2.0.0 is up
```

If you do not see this message, then Oracle Loader for Hadoop is not installed properly. Check that these environment variables are set correctly:

- `OLH_HOME`: Set to the installation directory
- `HADOOP_CLASSPATH`: Includes $`OLH_HOME`/jlib/*
- `CLASSPATH`: Includes $`OLH_HOME`/jlib/*

If `hdfs.pull` fails and `HADOOP_CLASSPATH` is set correctly, then the version of Oracle Loader for Hadoop may be incorrect for the version of CDH. Check the Oracle Loader for Hadoop log file.

> **See Also:** "Oracle Loader for Hadoop Setup" on page 1-12 for installation instructions

Return Value

An `ore.frame` object that points to the database table with data loaded from HDFS, or `NULL` if the operation failed

See Also

Oracle R Enterprise User's Guide for a description of `ore.frame` objects.

hdfs.push

Copies data from an Oracle database to HDFS.

This operation requires authentication by Oracle Database. See orch.connect on page 6-41.

Usage

```
hdfs.push(
        x,
        key,
        dfs.name,
        overwrite,
        driver,
        split.by)
```

Arguments

x
An `ore.frame` object with the data in an Oracle database to be pushed.

key
The index or name of the key column.

dfs.name
Unique name for the object in HDFS.

overwrite
`TRUE` to allow `dfs.name` to overwrite an object with the same name, or `FALSE` to signal an error (default).

driver
Identifies the driver used to copy the data. This argument is currently ignored because Sqoop is the only supported driver.

split.by
The column to use for data partitioning (required).

Usage Notes

Because this operation is synchronous, copying a large data set may take a while. The prompt reappears and you regain use of R when copying is complete.

An `ore.frame` object is an Oracle R Enterprise metadata object that points to a database table. It corresponds to an R `data.frame` object.

If you omit the `split.by` argument, then `hdfs.push` might import only a portion of the data into HDFS.

Return Value

The full path to the file that contains the data set, or `NULL` if the operation failed

See Also

Oracle R Enterprise User's Guide

Example

This example creates an `ore.frame` object named `ontime_s2000` that contains the rows from the `ONTIME_S` database table in where the year equals 2000. Then `hdfs.push` uses `ontime_s2000` to create /user/oracle/xq/ontime2000_DB in HDFS.

```
R> ontime_s2000 <- ONTIME_S[ONTIME_S$YEAR == 2000,]
R> class(ontime_s2000)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> ontime2000.dfs <- hdfs.push(ontime_s2000, key='DEST', dfs.name='ontime2000_DB',
’split.by=’YEAR’)
R> ontime2000.dfs
[1] "/user/oracle/xq/ontime2000_DB"
attr(,"dfs.id")
[1] TRUE
```

hdfs.put

Copies data from an R in-memory object (data.frame) to HDFS. All data attributes, like column names and data types, are stored as metadata with the data.

Usage

```
hdfs.put(
        data,
        key,
        dfs.name,
        overwrite,
        rownames)
```

Arguments

data
An `ore.frame` object in the local R environment to be copied to HDFS.

key
The index or name of the key column.

dfs.name
A unique name for the new file.

overwrite
Controls whether `dfs.name` can overwrite a file with the same name. Set to `TRUE` to overwrite the file, or `FALSE` to signal an error.

rownames
Set to `TRUE` to add a sequential number to the beginning of each line of the file, or `FALSE` otherwise.

Usage Notes

You can use `hdfs.put` instead of `hdfs.push` to copy data from `ore.frame` objects, such as database tables, to HDFS. The table must be small enough to fit in R memory; otherwise, the function fails. The `hdfs.put` function first reads all table data into local R memory and then transfers it to HDFS. For a small table, this function can be faster than `hdfs.push` because it does not use Sqoop and thus does not have the overhead incurred by `hdfs.push`.

Return Value

The object ID of the new file, or `NULL` if the operation failed

Example

This example creates a file named /user/oracle/xq/testdata.dat with the contents of the `dat` data frame.

```
R> myfile <- hdfs.put(dat, key='DEST', dfs.name='testdata.dat')
R> print(myfile)
[1] "/user/oracle/xq/testdata.dat"
attr(,"dfs.id")
[1] TRUE
```

hdfs.pwd

Identifies the current working directory in HDFS.

Usage

```
hdfs.pwd()
```

Return Value

The current working directory, or `NULL` if your R environment is not connected to HDFS

Example

This example shows that /user/oracle is the current working directory.

```
R> hdfs.pwd()
[1] "/user/oracle/"
```

hdfs.rm

Removes a file or directory from HDFS.

Usage

```
hdfs.rm(
        dfs.id,
        force)
```

Arguments

dfs.id
The name of a file or directory in HDFS. The name can include a path that is either absolute or relative to the current path.

force
Controls whether a directory that contains files is deleted. Set to `TRUE` to delete the directory and all its files, or `FALSE` to cancel the operation (default).

Usage Notes

All object identifiers in Hadoop pointing to this data are invalid after this operation.

Return Value

`TRUE` if the data is deleted, or `FALSE` if the operation failed

Example

This example removes the file named data1.log in the current working HDFS directory:

```
R> hdfs.rm("data1.log")
[1] TRUE
```

hdfs.rmdir

Deletes a directory in HDFS.

Usage

```
hdfs.rmdir(
        dfs.name,
        force)
```

Arguments

dfs.name
Name of the directory in HDFS to delete. The directory can be an absolute path or relative to the current working directory.

force
Controls whether a directory that contains files is deleted. Set to `TRUE` to delete the directory and all its files, or `FALSE` to cancel the operation (default).

Usage Notes

This function deletes all data objects stored in the directory, which invalidates all associated object identifiers in HDFS.

Return Value

`TRUE` if the directory is deleted successfully, or `FALSE` if the operation fails

Example

```
R> hdfs.rmdir("mydata")
[1] TRUE
```

hdfs.root

Returns the HDFS root directory.

Usage

```
hdfs.root()
```

Return Value

A data frame with the full path of the HDFS root directory

Usage Notes

The default HDFS root is set to /. Users to not have write privileges to this directory. To change the HDFS root to a path where you have write access, use the hdfs.setroot function.

See Also

hdfs.setroot on page 6-35

Example

This example identifies /user/oracle as the root directory of HDFS.

```
R> hdfs.root()
[1] "/user/oracle"
```

hdfs.sample

Copies a random sample of data from a Hadoop file into an R in-memory object. Use this function to copy a small sample of the original HDFS data for developing the R calculation that you ultimately want to execute on the entire HDFS data set on the Hadoop cluster.

Usage

```
hdfs.sample(
        dfs.id,
        lines,
        sep)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

lines
The number of lines to return as a sample. The default value is 1000 lines.

sep
The symbol used to separate fields in the Hadoop file. A comma (`,`) is the default separator.

Usage Notes

If the data originated in an R environment, then all metadata is extracted and all attributes are restored, including column names and data types. Otherwise, generic attribute names, like `val1` and `val2`, are assigned.

This function can become slow when processing large input HDFS files, as the result of inherited limitations in the Hadoop command-line interface.

Return Value

A `data.frame` object with the sample data set, or `NULL` if the operation failed

Example

This example displays the first three lines of the ontime_R file.

```
R> hdfs.sample("ontime_R", lines=3)
  YEAR MONTH MONTH2 DAYOFMONTH DAYOFMONTH2 DAYOFWEEK DEPTIME...
1 2000    12     NA         31          NA         7     1730...
2 2000    12     NA         31          NA         7     1752...
3 2000    12     NA         31          NA         7     1803...
```

hdfs.setroot

Sets the HDFS root directory.

Usage

```
hdfs.setroot(dfs.root)
```

Arguments

dfs.root
The full path in HDFS to be set as the current root directory. If this argument is omitted, then your HDFS home directory is used as the root.

Usage Notes

Use hdfs.root on page 6-33 to see the current root directory.

All HDFS paths and operations using Oracle R Connector for Hadoop are relative to the current HDFS root. You cannot change the current working directory above the root directory. Users do not have write access to the HDFS / directory, which is the default HDFS root directory.

Return Value

The full path to the newly set HDFS root directory, or NULL if an error prevented a new root directory from being set

Example

This example changes the HDFS root directory from /user/oracle to /user/oracle/demo.

```
R> hdfs.root()
[1] "/user/oracle"
R> hdfs.setroot("/user/oracle/demo")
R> hdfs.root()
[1] "/user/oracle/demo"
```

hdfs.size

Returns the size of a file in HDFS.

Usage

```
hdfs.size(
        dfs.id,
        units)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

units
Specifies a unit of measurement for the return value:

- `KB` (kilobytes)
- `MB` (megabytes)
- `GB` (gigabytes)
- `TB` (terabytes)
- `PB` (petabytes)

The unit defaults to bytes if you omit the argument or enter an unknown value.

Usage Notes

Use this interface to determine, for instance, whether you can copy the contents of an entire HDFS file into local R memory or a local file, or if you can only sample the data while creating a prototype of your R calculation.

Return Value

Size of the object, or `0` if the object does not exist in HDFS

Example

This example returns a file size for ontime_R of 999,839 bytes.

```
R> hdfs.size("ontime_R")
[1] 999839
```

hdfs.tail

Copies a specified number of lines from the end of a file in HDFS.

Usage

```
hdfs.tail(
        dfs.id,
        n)
```

Arguments

dfs.id
The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

n
The number of lines to retrieve from the file.

Return Value

The last *n* lines of the file.

See Also

hdfs.head on page 6-19

Example

This example returns the last three lines of ontime.R:

```
R> hdfs.tail('ontime_R', 3)
[1]
"\"219930\",1994,8,\"M08\",1,\"D01\",1,1947,1945,2038,2048,\"US\",1073,NA,51,63,NA
,-10,2,\"PIT\",\"LAN\",275,NA,NA,0,NA,\"0\""
[2]
"\"219931\",1994,8,\"M08\",2,\"D02\",2,802,805,948,950,\"AA\",1726,NA,106,105,NA,-
2,-3,\"DFW\",\"BNA\",631,NA,NA,0,NA,\"0\""
[3]
"\"219932\",1994,8,\"M08\",2,\"D02\",2,829,832,1015,1011,\"US\",1191,NA,166,159,NA
,4,-3,\"BWI\",\"MSY\",998,NA,NA,0,NA,\"0\""
```

hdfs.upload

Copies a file from the local file system into HDFS.

Usage

```
hdfs.upload(
        filename,
        dfs.name,
        overwrite,
        split.size,
        header)
```

Arguments

filename
Name of a file in the local file system.

dfs.name
Name of the new directory in HDFS.

overwrite
Controls whether `dfs.name` can overwrite a directory with the same name. Set to `TRUE` to overwrite the directory, or `FALSE` to signal an error (default).

split.size
Maximum number of bytes in each part of the Hadoop file (optional).

header
Indicates whether the first line of the local file is a header containing column names. Set to `TRUE` if it has a header, or `FALSE` if it does not (default).

A header enables you to exact the column names and reference the data fields by name instead of by index in your MapReduce R scripts.

Usage Notes

This function provides the fastest and easiest way to copy a file into HDFS. If the file is larger than *`split.size`*, then Hadoop splits it into two or more parts. The new Hadoop file gets a unique object ID, and each part is named `part-0000x`. Hadoop automatically creates metadata for the file.

Return Value

HDFS object ID for the loaded data, or `NULL` if the copy failed

See Also

`hdfs.download` on page 6-16, `hdfs.get` on page 6-18, `hdfs.put` on page 6-29

Example

This example uploads a file named ontime_s2000.dat into HDFS and shows the location of the file, which is stored in a variable named `ontime.dfs_File`.

```
R> ontime.dfs_File <- hdfs.upload('ontime_s2000.dat', dfs.name='ontime_File')
R> print(ontime.dfs_File)
```

```
[1] "/user/oracle/xq/ontime_File"
```

is.hdfs.id

Indicates whether an R object contains a valid HDFS file identifier.

Usage

```
is.hdfs.id(x)
```

Arguments

x
The name of an R object.

Return Value

`TRUE` if x is a valid HDFS identifier, or `FALSE` if it is not

See Also

`hdfs.attach` on page 6-11, `hdfs.id` on page 6-20

Example

This example shows that `dfs` contains a valid HDFS identifier, which was returned by `hdfs.attach`:

```
R> dfs <- hdfs.attach('ontime_R')
R> is.hdfs.id(dfs)
[1] TRUE
R> print(dfs)
[1] "/user/oracle/xq/ontime_R"
attr(,"dfs.id")
[1] TRUE
```

The next example shows that a valid file name passed as a string is not recognized as a valid file identifier:

```
R> is.hdfs.id('/user/oracle/xq/ontime_R')
[1] FALSE
```

orch.connect

Establishes a connection to Oracle Database.

Usage

```
orch.connect(
        host,
        user,
        sid,
        passwd,
        port,
        secure,
        driver,
        silent)
```

Arguments

host
Host name or IP address of the server where Oracle Database is running.

user
Database user name.

sid
System ID (SID) for the Oracle Database instance.

passwd
Password for the database user. If you omit the password, you are prompted for it.

port
Port number for the Oracle Database listener. The default value is 1521.

secure
Authentication setting for Oracle Database:

- `TRUE`: You must enter a database password each time you attempt to connect (default).
- `FALSE`: You must enter a database password only once during a session. The encrypted password is kept in memory and used for subsequent connection attempts.

driver
Driver used to connect to Oracle Database (optional). Sqoop is the default driver.

silent
`TRUE` to suppress the prompts for missing host, user, password, port, and SID values, or `FALSE` to see them (default).

Usage Notes

Use this function when your analysis requires access to data stored in an Oracle database or to return the results to the database.

With an Oracle Advanced Analytics license for Oracle R Enterprise and a connection to Oracle Database, you can work directly with the data stored in database tables and pass processed data frames to R calculations on Hadoop.

You can reconnect to Oracle Database using the connection object returned by the `orch.dbcon` function.

Return Value

`TRUE` for a successful and validated connection, or `FALSE` for a failed connection attempt

See Also

`orch.dbcon` on page 6-45, `orch.disconnect` on page 6-52

Example

This example installs the `ORCH` library and connects to Oracle Database on the local system:

```
R> library(ORCH)
Loading required package: OREbase

Attaching package: OREbase

The following object(s) are masked from package:base:

    cbind, data.frame, eval, interaction, order, paste, pmax, pmin,
    rbind, table

Loading required package: OREstats
Loading required package: MASS
Loading required package: ORCHcore
Oracle R Connector for Hadoop 2.0
Hadoop 2.0.0-cdh4.1.2 is up
Sqoop 1.4.1-cdh4.1.2 is up
OLH 2.0.0 is up

R> orch.connect("localhost", "RQUSER", "orcl")
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected.
[1] TRUE
```

> **Note:** If you see the message "p with Hadoop MapReduce" when you load `ORCH`, then clear the `HADOOP_HOME` environment variable. See "Installing the Software on Hadoop" on page 1-14.

The next example uses a connection object to reconnect to Oracle Database:

```
R> conn<-orch.dbcon()
R> orch.disconnect()
Disconnected from a database.
```

```
R> orch.connect(conn)
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE
```

orch.connected

Checks whether Oracle R Connector for Hadoop is connected to Oracle Database.

Usage

```
orch.connected()
```

Return Value

`TRUE` if Oracle R Connector for Hadoop has a connection to Oracle Database, or `FALSE` if it does not.

Example

This example shows that Oracle R Connector for Hadoop does not have a connection to Oracle Database:

```
R> orch.connected()
[1] TRUE
```

orch.dbcon

Returns a connection object for the current connection to Oracle Database, excluding the authentication credentials.

Usage

```
orch.dbcon()
```

Return Value

A data frame with the connection settings for Oracle Database

Usage Notes

Use the connection object returned by `orch.dbcon` to reconnect to Oracle Database using `orch.connect`.

See Also

orch.connect on page 6-41

Example

This example shows how you can reconnect to Oracle Database using the connection object returned by `orch.dbcon`:

```
R> orch.connect('localhost', 'RQUSER', 'orcl')
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE

R> conn<-orch.dbcon()
R> orch.disconnect()
Disconnected from a database.

R> orch.connect(conn)
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE
```

The following shows the connection object returned by `orch.dbcon` in the previous example:

```
R> conn
Object of class "orch.dbcon"
data frame with 0 columns and 0 rows
Slot "ok":
```

```
[1] TRUE

Slot "host":
[1] "localhost"

Slot "port":
[1] 1521

Slot "sid"
[1] "orcl"

Slot "user":
[1] "RQUSER"

Slot "passwd":
[1] ""

Slot "secure":
[1] TRUE

Slot "drv":
[1] "sqoop"
```

orch.dbg.lasterr

Returns the last error message.

Usage

```
orch.dbg.lasterr()
```

Return Value

The last error message.

Example

This example returns the last error message, which indicates a problem with the Hadoop installation.

```
R> orch.dbg.lasterr()
[1] "split column must be specified"
```

orch.dbg.off

Turns off debugging mode.

Usage

```
orch.dbg.off()
```

Return Value

```
FALSE
```

See Also

orch.dbg.on on page 6-49

Example

This example turns off debugging:

```
R> orch.dbg.off()
```

orch.dbg.on

Turns on debugging mode.

Usage

```
orch.dbg.on(severity)
```

Arguments

severity
Identifies the type of error messages that are displayed. You can identify the severity by the number or by the case-insensitive keyword shown in Table 6–1.

Table 6–1 Debugging Severity Levels

Keyword	Number	Description
`all`	11	Return all messages.
`critical`	1	Return only critical errors.
`error`	2	Return all errors.
`warning`	3	Return all warnings.

Return Value

The severity level

See Also

`orch.dbg.output` on page 6-50, `orch.dbg.off` on page 6-48

Example

This example turns on debugging for all errors:

```
R> severe<-orch.dbg.on(severity<-2)
R> severe
[1] "ERROR" "2"
```

orch.dbg.output

Directs the output from the debugger.

Usage

```
orch.dbg.output(con)
```

Arguments

con
Identifies the stream where the debugging information is sent: `stderr()`, `stdout()`, or a file name.

Usage Notes

You must first turn on debugging mode before redirecting the output.

Return Value

The current stream

See Also

`orch.dbg.on` on page 6-49

Example

This example turns on debugging mode and sends the debugging information to `stderr`. The `orch.dbg.output` function returns a description of `stderr`.

```
R> orch.dbg.on('all')
R> err<-orch.dbg.output(stderr())
17:32:11 [SY] debug output set to "stderr"
R> print(err)
description          class     mode      text    opened    can read    can write
   "stderr"     "terminal"      "w"    "text"   "opened"       "no"        "yes"
```

The next example redirects the output to a file named debug.log:

```
R> err<-orch.dbg.output('debug.log')
17:37:45 [SY] debug output set to "debug.log"
R> print(err)
[1] "debug.log"
```

orch.dbinfo

Displays information about the current connection.

Usage

```
orch.dbinfo(dbcon)
```

Arguments

dbcon
An Oracle Database connection object.

Return Value

```
NULL
```

See Also

orch.dbcon on page 6-45

Example

This example displays the information stored in a connection object named `conn`.

```
R> orch.dbinfo(conn)
Connected to a database via [sqoop]
    Host: localhost
    Port: 1521
    SID:  orcl
    User: RQUSER
```

orch.disconnect

Disconnects the local R session from Oracle Database.

Usage

```
orch.disconnect(
        silent,
        dbcon)
```

Arguments

silent
Set to `TRUE` to suppress all messages, or `FALSE` to see them (default).

dbcon
Set to `TRUE` to display the connection details, or `FALSE` to suppress this information (default).

Usage Notes

No `orch` functions work without a connection to Oracle Database.

Return Value

An Oracle Database connection object when `dbcon` is `TRUE`, otherwise `NULL`

See Also

`orch.connect` on page 6-41

Example

This example disconnects the local R session from Oracle Database:

```
R> orch.disconnect()
Disconnected from a database.
```

The next example disconnects the local R session from Oracle Database and displays the returned connection object:

```
R> oid<-orch.disconnect(silent=TRUE, dbcon=TRUE)
R> oid
Object of class "orch.dbcon"
data frame with 0 columns and 0 rows
slow "ok":
[1] TRUE

Slot "host":
[1] "localhost"

Slow "port":
[1] 1521

Slot "sid":
[1] orcl

Slot "user":
```

```
[1] RQUSER

Slot "passwd":
[1] ""

Slot "secure":
[1] TRUE

Slot "drv":
[1] "sqoop"
```

orch.dryrun

Switches the execution platform between the local host and the Hadoop cluster. No changes in the R code are required for a dry run.

Usage

```
orch.dryrun(onoff)
```

Arguments

onoff
Set to `TRUE` to run a MapReduce program locally, or `FALSE` to run the program on the Hadoop cluster.

Usage Notes

The `orch.dryrun` function enables you to run a MapReduce program locally on a laptop using a small data set before running it on a Hadoop cluster using a very large data set. The mappers and reducers are run sequentially on row streams from HDFS. The Hadoop cluster is not required for a dry run.

Return Value

The current setting of `orch.dryrun`

See Also

`hadoop.exec` on page 6-5, `hadoop.run` on page 6-8

Example

This example changes the value of `orch.dryrun` from `FALSE` to `TRUE`.

```
R> orch.dryrun()
[1] FALSE
R> orch.dryrun(onoff<-T)
R> orch.dryrun()
[1] TRUE
```

orch.export

Makes R objects from a user's local R session available in the Hadoop execution environment, so that they can be referenced in MapReduce jobs.

Usage

```
orch.export(...)
```

Arguments

...
One or more variables, data frames, or other in-memory objects, by name or as an explicit definition, in a comma-separated list.

Usage Notes

You can use this function to prepare local variables for use in `hadoop.exec` and `hadoop.run` functions. The `mapper`, `reducer`, `combiner`, `init`, and `final` arguments can reference the exported variables.

Return Value

A list object

See Also

`hadoop.exec` on page 6-5, `hadoop.run` on page 6-8

Example

This code fragment shows `orch.export` used in the `export` argument of the `hadoop.run` function:

```
hadoop.run(x,
    export = orch.export(a=1, b=2),
    mapper = function(k,v) {
        x <- a + b
        orch.keyval(key=NULL, val=x)
    }
)
```

The following is a similar code fragment, except that the variables are defined outside the `hadoop.run` function:

```
a=1
b=2
hadoop.run(x,
    export = orch.export(a, b),
    .
    .
    .
)
```

orch.keyval

Outputs key-value pairs in a MapReduce job.

Usage

```
orch.keyval(
        key,
        value)
```

Arguments

key
A scalar value.

value
A data structure such as a scalar, list, data frame, or vector.

Usage Notes

This function can only be used in the `mapper`, `reducer`, or `combiner` arguments of `hadoop.exec` and `hadoop.run`. Because the `orch.keyval` function is not exposed in the `ORCH` client API, you cannot call it anywhere else.

Return Value

(*key*, *value*) structures

See Also

`orch.pack` on page 6-59

Example

This code fragment creates a mapper function using `orch.keyval`:

```
hadoop.run(data,
     mapper = function(k,v) {
        orch.keyval(k,v)
     })
```

orch.keyvals

Outputs a set of key-value pairs in a MapReduce job.

Usage

```
orch.keyvals(
        key,
        value)
```

Arguments

key
A scalar value.

value
A data structure such as a scalar, list, data frame, or vector.

Usage Notes

This function can only be used in the `mapper`, `reducer`, or `combiner` arguments of `hadoop.exec` and `hadoop.run`. Because the `orch.keyvals` function is not exposed in the `ORCH` client API, you cannot call it anywhere else.

See Also

`orch.keyval` on page 6-56, `orch.pack` on page 6-59

Return Value

(*key*, *value*) structures

Example

This code fragment creates a mapper function using `orch.keyval` and a reducer function using `orch.keyvals`:

```
hadoop.run(data,
     mapper(k,v) {
          if (v$value > 10) {
               orch.keyval(k, v)
          }
          else {
               NULL
          }
     },
     reducer(k,vals) {
          orch.keyvals(k,vals)
}
)
```

The following code fragment shows `orch.keyval` in a `for` loop to perform the same reduce operation as `orch.keyvals` in the previous example:

```
     reducer(k,vals) {
          out <- list()
          for (v in vals) {
```

```
            out <- list(out, orch.keyval(k,vals))
        }
        out
}
```

orch.pack

Compresses one or more in-memory R objects that the mappers or reducers must write as the values in key-value pairs.

Usage

```
orch.pack(...)
```

Arguments

...
One or more variables, data frames, or other in-memory objects in a comma-separated list.

Usage Notes

You should use this function when passing nonscalar or complex R objects, such as data frames and R classes, between the mapper and reducer functions. You do not need to use it on scalar or other simple objects. You can use `orch.pack` to vary the data formats, data sets, and variable names for each output value.

You should also use `orch.pack` when storing the resultant data set in HDFS. The compressed data set is not corrupted by being stored in an HDFS file.

The `orch.pack` function must always be followed by the `orch.unpack` function to restore the data to a usable format.

Return Value

Compressed character-type data as a long string with no special characters

See Also

`hadoop.exec` on page 6-5, `hadoop.run` on page 6-8, `orch.keyval` on page 6-56, `orch.unpack` on page 6-62

Example

This code fragment compresses the content of several R objects into a serialized stream using `orch.pack`, and then creates key-value pairs using `orch.keyval`:

```
orch.keyval(NULL, orch.pack(
     r = r,
     qy = qy,
     yy = yy,
     nRows = nRows))
```

orch.reconnect

Reconnects to Oracle Database with the credentials previously returned by `orch.disconnect`.

> **Note:** The `orch.reconnect` function is deprecated in release 1.1 and will be desupported in release 2.0. Use `orch.connect` to reconnect using a connection object returned by `orch.dbcon`.

Usage

```
orch.reconnect(dbcon)
```

Arguments

dbcon
Credentials previously returned by `orch.disconnect`.

Usage Notes

Oracle R Connector for Hadoop preserves all user credentials and connection attributes, enabling you to reconnect to a previously disconnected session. Depending on the `orch.connect` secure setting for the original connection, you may be prompted for a password. After reconnecting, you can continue data transfer operations between Oracle Database and HDFS.

Reconnecting to a session is faster than opening a new one, because reconnecting does not require extensive connectivity checks.

Return Value

`TRUE` for a successfully reestablished and validated connection, or `FALSE` for a failed attempt

See Also

`orch.connect` on page 6-41

Example

```
R> orch.reconnect(oid)
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID:  orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE
```

orch.temp.path

Sets the path where temporary data is stored.

Usage

```
orch.temp.path(path)
```

Arguments

path
The full path to an existing HDFS directory.

Return Value

The current temporary path

Example

This example returns /tmp as the current temporary directory.

```
R> orch.temp.path()
[1] "/tmp"
```

orch.unpack

Restores the R objects that were compressed with a previous call to `orch.pack`.

Usage

```
orch.unpack(...)
```

Arguments

...
The name of a compressed object.

Usage Notes

This function is typically used at the beginning of a mapper or reducer function to obtain and prepare the input. However, it can also be used externally, such as at the R console, to unpack the results of a MapReduce job.

Consider this data flow:

`ORCH` client to mapper to combiner to reducer to `ORCH` client.

If the data is packed at one stage, then it must be unpacked at the next stage.

Return Value

Uncompressed list-type data, which can contain any number and any type of variables

See Also

`orch.pack` on page 6-59

Example

This code fragment restores the data that was compressed in the `orch.pack` example:

```
reducer = function(key, vals) {
     x    <-orch.unpack(vals[[1]])
     r    <-x$qy
     yy   <- x$yy
     nRow <- x$nRows
     .
     .
     .
     }
```

orch.version

Identifies the version of the ORCH package.

Usage

```
orch.version()
```

Return Value

ORCH package version number

Example

This example shows that the ORCH version number is 2.0.

```
R> orch.version()
[1] "2.0"
```

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X